Н. Г. ДРУЖИНИНА, О. Г. ТРОФИМОВА

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

Н. Г. Дружинина, О. Г. Трофимова

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Рекомендовано методическим советом УрФУ
в качестве учебно-методического пособия для студентов,
обучающихся по программе бакалавриата по направлению подготовки
220400 – Управление в технических системах

Екатеринбург
УрФУ
2012

УДК 004.94(075.8)

ББК 32.973.26-018я73
Д76

Рецензенты:

кафедры прикладной информатики в социальных коммуникациях и информационных технологий института урбанистики Уральской государственной архитектурной академии (декан факультета прикладной информатики, завкафедрой «Прикладная информатика в социальных коммуникациях» доц., канд. тех. наук Г. Б. Захарова, доц. каф. «Информационные технологии» доц., канд. физ.-мат. наук А. И. Кривоногов); начальник службы автоматики и связи МУП Трамвайно-троллейбусного управления г. Екатеринбург Н. П. Дроздов

Научный редактор доц., канд. техн. наук В. А. Морозова

Дружинина, Н. Г.

Д76 ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ : учебно-методическое пособие / Н. Г. Дружинина, О. Г. Трофимова. – Екатеринбург : УРФУ, 2012. – 144 с.

ISBN 978-5-321-02202-3

Лабораторный практикум по дисциплине «Информационное обеспечение систем управления» предназначен для студентов. Представлены основные возможности популярной инструментальной среды разработки приложений *Delphi* и языка программирования Интернет-приложений *PHP*. Описаны ключевые моменты объектно-ориентированного программирования. Приведены простые примеры разработки приложений в среде *Delphi,* веб-сайта с помощью *PHP*. Рассмотрены возможности языка *MySQL*, реализованные в СУБД *MySQL*. Основной акцент сделан на разработке приложений с использованием баз данных.
Библиогр. : 11 назв. Табл. 11. Рис. 57.

УДК 001.57+004.942(075.8)

ББК 32.973.26-018я73

ISBN 978-5-321-02202-3

1. СРЕДСТВА И ПРИЕМЫ СОЗДАНИЯ ПРИЛОЖЕНИЙ В СРЕДЕ *DELPHI*

1.1. Среда *Delphi*

Delphi представляет собой не только инструмент разработки приложений, но и сложную среду программирования. Создание прикладных программ, или приложений, *Delphi* осуществляется в интегрированной среде разработки *IDE* (*Integrated Development Environment*). Интегрированная среда разработки *Delphi* представляет собой многооконную систему. После загрузки интерфейс *Delphi* выглядит так, как показано на рис. 1.

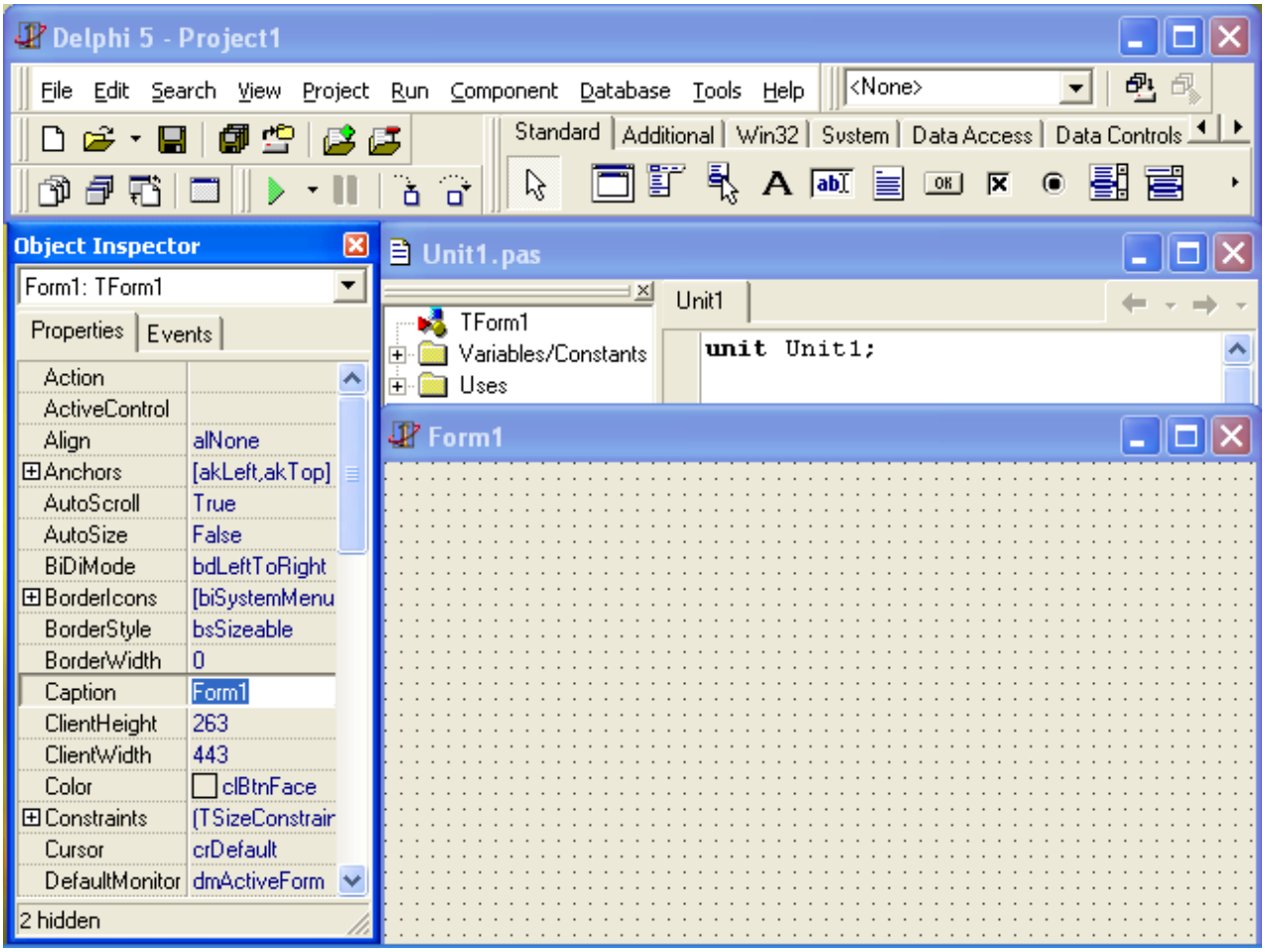

Рис. 1. Вид интегрированной среды разработки

Интерфейс *Delphi* первоначально имеет четыре окна:

- главное окно (*Delphi – Project*1);
- окно *Инспектор объектов* (*Object Inspector*);

- окно *Конструктор формы* (*Form*1);
- окно *Редактор кода* (*Unit1.pas*).

В главном окне *Delphi* отображаются:

- главное меню;
- панели инструментов;
- палитра инструментов.

Главное меню содержит обширный набор команд для доступа к функциям *Delphi*.

Панель инструментов находится под главным меню в левой части главного окна и содержит 15 кнопок для вызова часто используемых команд главного меню, например, *File/Open* (*Файл/Открыть*) или *Run/Run* (*Выполнение/Выполнить*).

Всего имеется 5 панелей:

- *Standard* (*Стандартная*);
- *View* (*Просмотр*);
- *Debug* (*Отладка*);
- *Custom* (*Пользователь*);
- *Desktop* (*Рабочий стол*).

Палитра компонентов находится под главным меню в правой части главного окна и содержит множество компонентов, размещаемых в создаваемых формах. Компоненты являются своего рода строительными блоками, из которых конструируются формы приложения. Все компоненты располагаются на отдельной вкладке (странице), а сами компоненты представлены соответствующими значками (пиктограммами). Первоначально палитра компонентов имеет следующие вкладки:

- *Standard – Стандартная*;
- *Additional – Дополнительная*;
- *Win*32 – 32*-разрядный интерфейс Windows*;
- *System – Доступ к системным функциям*.

1.1.1. Характеристика проекта

Приложение, создаваемое в среде *Delphi*, состоит из нескольких элементов, объединенных в проект. В состав проекта входят следующие элементы (в скобках указаны расширения имен файлов):

- код проекта (*DPR*);
- описания форм (*DFM*);
- модули форм (*PAS*);
- модули (*PAS*);
- параметры проекта (*OPT*);
- описание ресурсов (*RES*).

Кроме приведенных файлов автоматически могут создаваться и другие, например их резервные копии: *~DP* – для *DPR*-файлов, *~PA* – для *PAS*-файлов.

1.1.2. Файл проекта

При запуске *Delphi* автоматически создается новый проект *Project*1, имя которого отображается в заголовке главного окна *Delphi*. Этот проект имеет в своем составе одну форму *Form*1.

Файл проекта является основным и представляет собой собственно программу. Для приложения, включающего в свой состав одну форму, файл проекта имеет следующий вид:

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
```

Application.Run;

end.

Имя программы совпадает с именем проекта и указывается при сохранении этого файла на диске.

Сборка всего проекта выполняется при компиляции файла проекта. В таком случае имя создаваемого приложения (*EXE*-файла) совпадает с названием файла проекта.

В разделе *Uses* указывается имя подключаемого модуля *Forms* и перечисляются подключаемые модули всех форм проекта, первоначально это модуль *Unit*1 формы *Form*1.

Директива *$R* подключает к проекту файл ресурсов, который по умолчанию имеет имя, совпадающее с именем файла проекта, поэтому вместо имени файла ресурса указан символ «*». Кроме этого файла разработчик может подключить к проекту и другие ресурсы, самостоятельно добавив директивы *$R* и указав в них соответствующие имена файлов ресурсов.

Программа проекта содержит всего три оператора, выполняющих инициализацию приложения, создание формы *Form*1 и запуск приложения. При выполнении разработчиком каких-либо операций с проектом код файла проекта формируется *Delphi* автоматически. Например, при добавлении новой формы в файл проекта добавляется две строки кода, относящиеся к этой форме.

Для просмотра и редактирования кода файла проекта в окне *Редактор кода* достаточно выполнить команду *Project/View Source* (*Проект/Просмотр источника*).

1.1.3. Форма проекта

Для каждой формы в составе проекта автоматически создается файл описания (*DFM*) и файл модуля (*PAS*). *Файл описания формы* содержит характеристики формы и ее компонентов. Разработчик обычно самостоятельно управляет этим файлом, используя окно *Конструктор формы* и *Инспектор объектов*. Содержимое файла описания формы определяет ее вид.

Оно доступно через *Конструктор формы*. При необходимости можно отобразить данный файл на экране в текстовом виде. Для этого нужно закрыть окно *Конструктора* той формы, для которой выполняется отображение файла описания, после чего по команде *File/Open* (*Файл/Открыть*) файл описания формы открывается в окне. Редактор кода и его содержимое доступно для просмотра и редактирования. Ниже приведен пример описания формы, на которой расположена кнопка *Button*1; кроме того, для этой кнопки создан разработчик события *OnClick* (*Нажатие*):

```
object Form1: TForm1
  Left = 192
  Top = 154
  Width = 451
  Height = 297
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Button1: TButton
    Left = 64
    Top = 40
    Width = 75
    Height = 25
    Caption = 'Button1'
    TabOrder = 0
```

OnClick = Button1Click

end

end.

Для того чтобы снова открыть окно *Конструктор формы*, предварительно в Редакторе кода командой меню *File/Close* (*Файл/Закрыть*) должен быть закрыт файл описания соответствующей формы. Открытие окна *Конструктор формы* выполняется командой *View/Form…* (*Просмотр формы*). Открывается диалоговое окно *View Form*, в котором можно выбрать нужную форму (рис. 2).

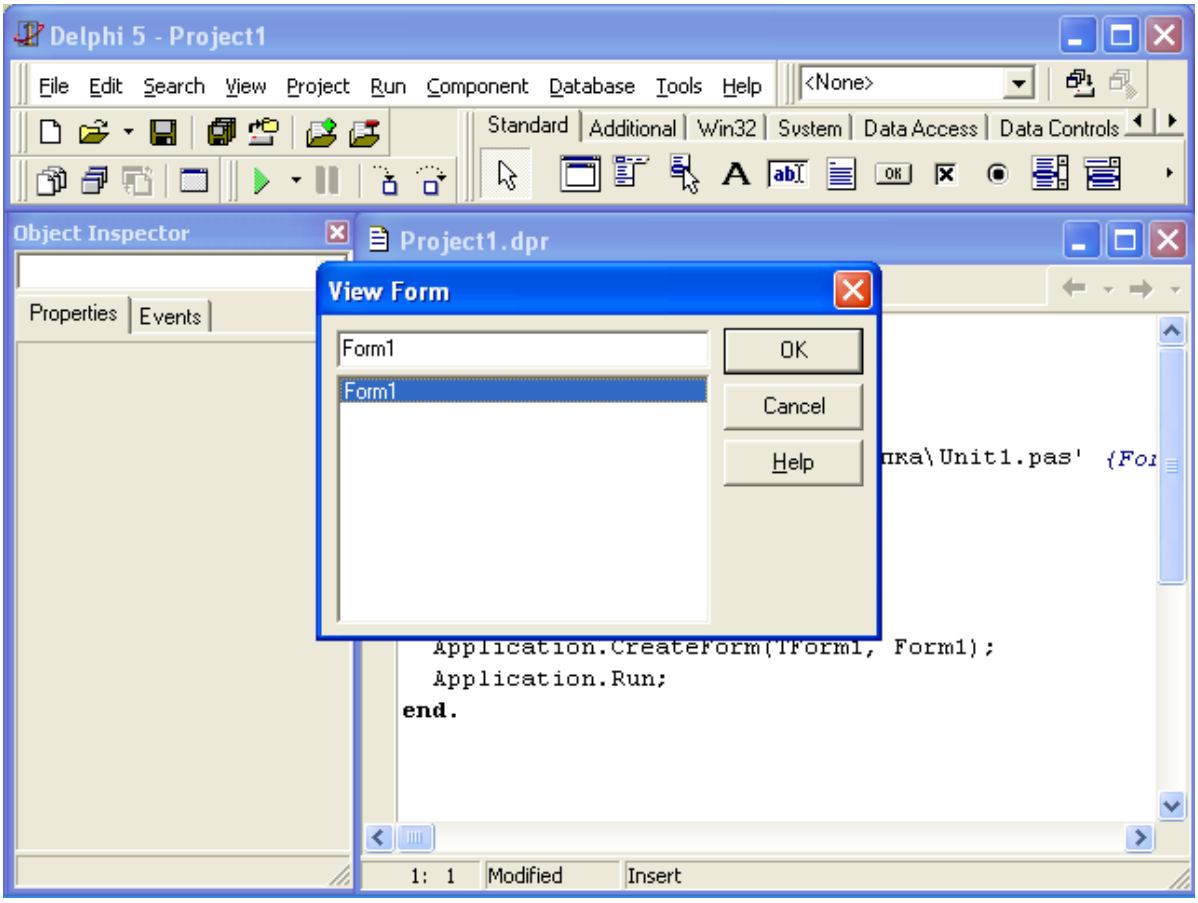

Рис. 2. Открытие окна *Конструктор формы*

Файл модуля формы содержит описание класса формы. Для пустой формы, добавляемой к проекту по умолчанию, файл модуля содержит следующий код:

***unit** Unit1;*

interface

uses

```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
end.
```

Файл модуля формы создается *Delphi* автоматически при добавлении новой формы. По умолчанию к проекту добавляется форма типа *TForm*, не содержащая компонентов.

В разделе *interface* модуля формы содержится описание класса формы, а в разделе *implementation* – подключение к модулю с помощью директивы *$R* описания соответствующей формы. При размещении на форме компонентов, а также при создании разработчиков событий в модуль вносятся соответствующие изменения.

Delphi относится к системам визуального программирования. В процессе разработки в форму помещают компоненты и для них устанавливаются необходимые свойства и обработчики событий.

1.1.4. Интуитивный помощник написания кода

Редактор кода *Delphi* оснащен набором средств, обеспечивающих выполнение целого ряда вспомогательных функций. Эти средства имеют общее название *Code Insight – Интуитивный помощник написания кода* (Insight – интуиция). *Code Insight* имеет пять составляющих:

- дополнение кода (*Code Completion*);
- контекстный список параметров (*Code Parameters*);
- быстрая оценка значения (*Tooltip Expression Evaluation*);
- всплывающая подсказка об объявлениях идентификаторов (*Tooltip Symbol Insight*);
- шаблоны кода (*Code Templates*).

Посредством функции дополнения кода в программу могут быть легко введены имена свойств, методов и обработчиков событий объектов. Сначала необходимо ввести имя объекта, поставив в конце точку, например: *Edit*1. После некоторой паузы *Delphi* отобразит на экране список всех свойств и методов данного объекта (рис. 3).

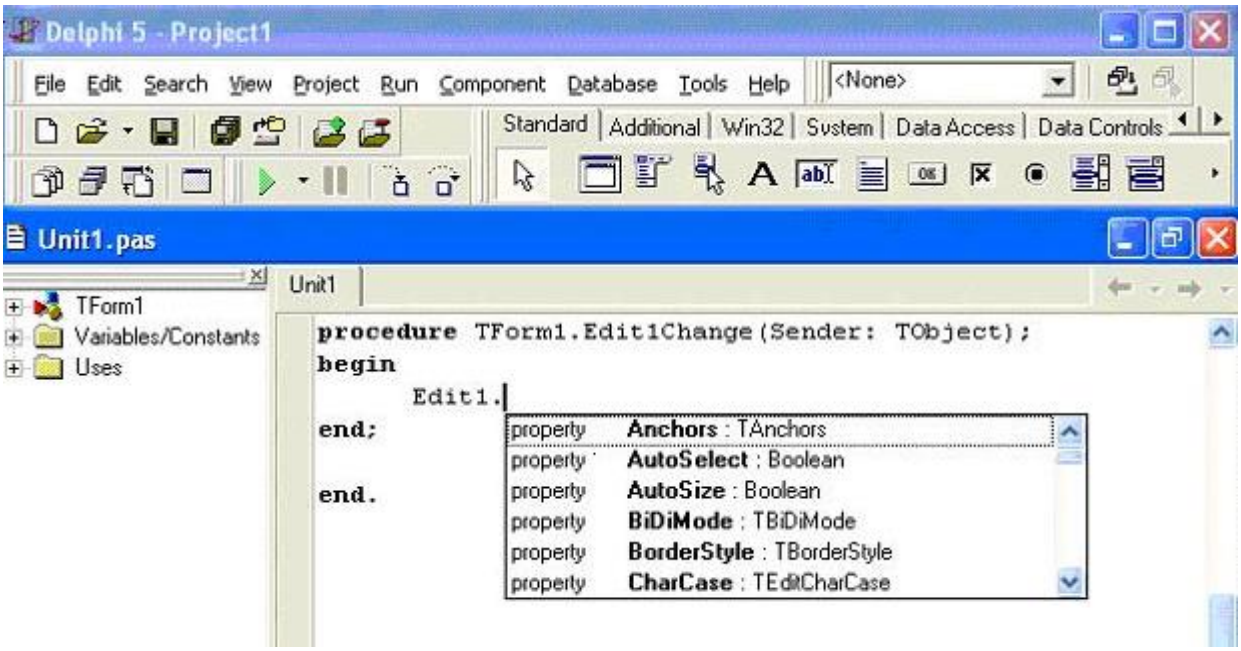

Рис. 3. Окно *Code Completion* в окне *Редактор кода*

Шаблоны кода содержат часто используемые программные конструкции, которые можно легко включить в программный код. Шаблоны кода активизируются нажатием клавиш [*Ctrl+J*]. Для целенаправленного поиска шаблона следует сначала ввести ключевое слово (например, *If* или *array*), а затем воспользоваться сочетанием клавиш [*Ctrl+J*]. После этого на экране отобразится список искомых шаблонов кода.

После ввода имени процедуры и открывающейся скобки функция контекстного списка параметров выводит на экране справочное окно *Hint* со списком параметров функции или метода.

Функция быстрой оценки кода позволяет проверить значения переменных и свойств. Если в режиме отладки приостановить выполнение приложения и установить курсор на имени переменной или свойства в окне редактора кода, через некоторое время на экране появится окно с текущим значением этой переменной.

Всплывающая подсказка об объявлениях идентификаторов отображает на экране всплывающую подсказку о типе и месте объявления идентификатора при установке курсора на его имени в окне редактора кода.

1.1.5. Инспектор объектов

При разработке приложения часто приходится использовать *Инспектор объектов* (*Object Inspector*). Окно *Object Inspector* содержит две страницы, каждую из них можно активизировать, выполнив щелчок на вкладке с соответствующим названием. Первая страница имеет название *Properties*. Левая колонка этой страницы отображает список всех свойств редактируемого компонента, доступных во время редактирования. Вторая колонка называется *Events*. В ее левой колонке перечислены все имеющиеся обработчики событий компонента. В правых колонках обеих страниц могут устанавливаться значения соответствующих свойств или обработчиков событий. На рис. 1 изображены страницы *Properties* и *Events* инспектора объектов для новой формы.

1.2. Примеры создания простых приложений и задания к лабораторной работе № 1

Цель лабораторной работы № 1 – познакомиться с основными средствами, компонентами, свойствами, методами и приемами создания приложений в среде *Delphi*. Используя представленные в примерах приемы разработки приложений, выполнить задания.

Пример 1

Создать новую форму. В форму поместить компоненты: кнопки *Button*1, *Button*2, *Edit*1, *Edit*2, *Edit*3, *ListBox*1.

Внешний вид компонента определяется его свойствами, которые доступны в окне *Инспектор объектов*, когда компонент выделен на форме и вокруг него отображаются маркеры выделения. Выберем свойство *Caption* и изменим заголовок кнопки *Закрыть*.

Для того чтобы кнопка могла реагировать на какое-либо событие, необходимо создать или указать процедуру обработки события, которая будет вызываться при возникновении данного события. Поскольку при нажатии на кнопку возникает событие *OnClick,* поскольку следует создать обработчик данного события. Для этого нужно в *Инспекторе объектов* (вкладка *Events*) сделать двойной щелчок в области значения события *OnClick* и написать код – *Close*.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   close;
end;
procedure TForm1.Button2Click(Sender: TObject);
var i:integer;
begin
    Edit1.Text:= IntToStr(Form1.ComponentCount);
    ListBox1.Clear;
    for i:=0 to   Form1.ComponentCount-1 do
    begin
        ListBox1.Items.add(IntToStr(i+1)+'.'+Form1.Components[i].Name);
    end;
end;
```

Для кнопки *Button*2 напишем программный код, выводящий список всех компонентов формы в список *ListBox*1. Результат выполнения события показан

на рис. 4. Используя компонент *OpenColor*1, можно изменить фон компонента *ListBox*1, цвет выводимого списка по выбору из предложенного списка компонента *OpenColor*1.

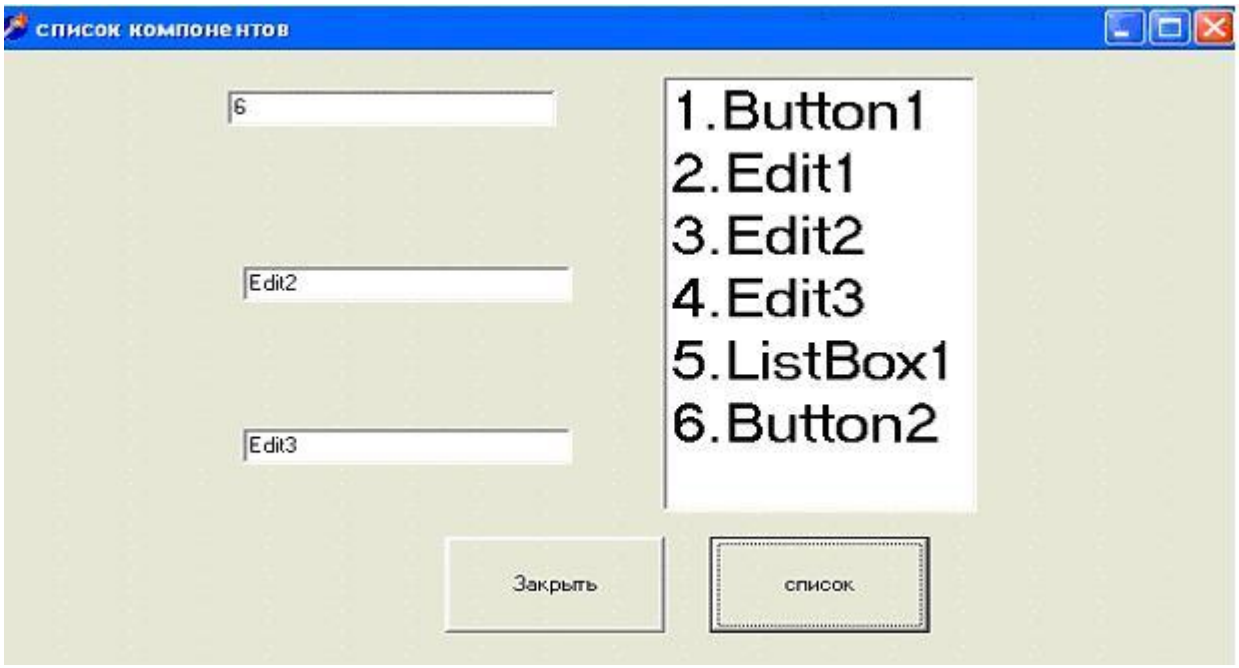

Рис. 4. Результаты процедуры *Button*2*Click* – заполненный список *TListBox*

Задание 1. Используя компонент *OpenColor1*, измените фон компонента *ListBox1*, цвет выводимого списка по выбору из предложенного списка компонента *OpenColor1*.

Пример 2

Компонент *TSplitter* – это компонент страницы *Additional* палитры компонентов. *TSplitter* позволяет изменять размеры элементов управления непосредственно во время выполнения приложения путем перетаскивания разделительных полос.

Для того чтобы использовать в приложении компонент *TSplitter*, необходимо:

- поместить элемент управления (например, *TMemo*) в форму и выровнять его по одной из сторон формы, присвоив свойству значение *AlLeft* или *AlRIght*;

- поместить в форму компонент *TSplitter* и присвоить его свойству *Align* то же значение, что и свойству *Align* вставленного ранее элемента управления;
- повторить два предыдущих действия всех элементов управления формы;
- присвоить свойству *Align* последнего элемента управления значение *AlClient* (чтобы этот элемент заполнял все оставшееся в форме место).

Для того чтобы элементы управления, помещенные в форму, можно было уменьшать до определенного размера, следует присвоить соответствующее значение свойству *MinSize*. После запуска проекта форма выглядит так, как показано на рис. 5. Используя свойства *alTop* и *alButtom*, можно сделать разделительные полосы вертикально.

Рис. 5. Форма с компонентами *TSplitter* после изменения размера

Задание 2. Используя свойства *alTop* и *alButtom*, сделайте разделительные полосы вертикально.

Пример 3

Расположите в форме компоненты группу опций *RadioGroup*1, линейку прокрутки *ScrollBar*1, кнопку *BitBtn*1. Для *BitBtn*1 у свойства *Kind* выберите *bkClose*. Это свойство определяет, какое изображение и/или текст будут отображаться на кнопке *BitBtn*. После выполнения щелчка на опции расположение линейки прокрутки меняется (рис. 6).

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    RadioGroup1.Items.add('Вертикально');
    RadioGroup1.Items.add('Горизонтально');
    RadioGroup1.ItemIndex:=2;
end;
```

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
    if RadioGroup1.Items[RadioGroup1.ItemIndex]= 'Вертикально'
    then
    ScrollBar1.Kind:=sbVertical;
    if RadioGroup1.Items[RadioGroup1.ItemIndex]= 'Горизонтально'
    then
    ScrollBar1.Kind:=sbHorizontal;
end.
```

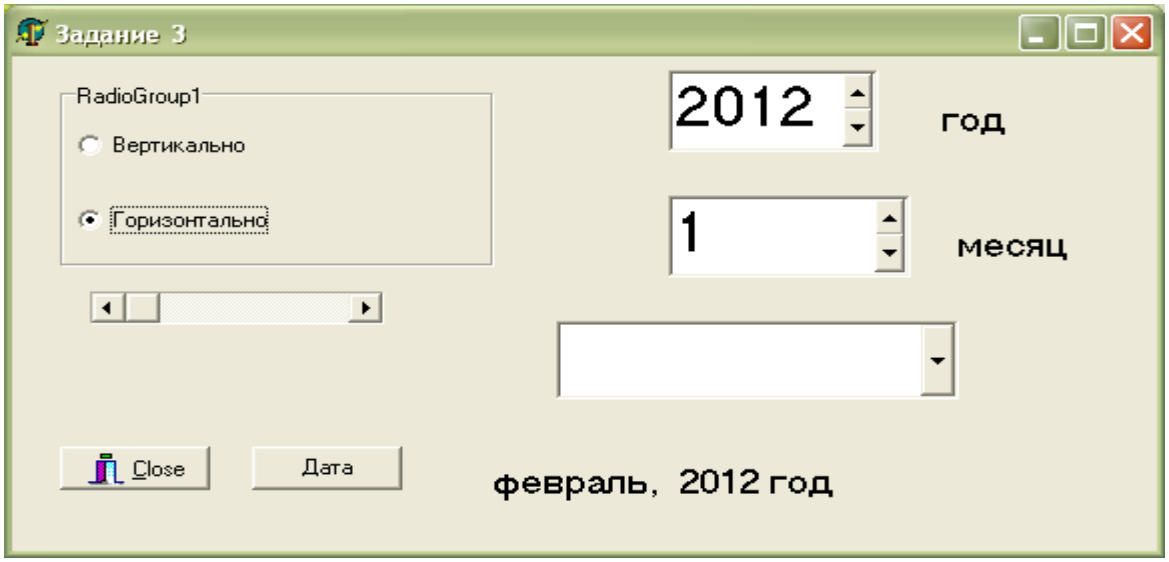

Рис. 6. Результат нажатия кнопки *Дата*

Если выполнить следующие действия: в форму поместить компоненты *SpinEdit*1, *SpinEdit*2, *Label*1, *Label*2, *ComboBox*1; свойству *Value* компонента *SpinEdit*1 присвоить начальное значение 2012; свойству *Value* компонента *SpinEdit*2 присвоить начальное значение 1; у *Label*1 подписать год, у *Label*2 – месяц; в *ComboBox*1 перечислить названия месяцев года; добавить кнопку; присвоить ей название – «Дата», – то по щелчку по этой кнопке в компоненте *Label*3 должно появиться название месяца, которое будет соответствовать значению переменной в компоненте *SpinEdit*2, и год, который будет соответствовать значению в компоненте *SpinEdit*1. Изменяя значения в компонентах *SpinEdit*1, *SpinEdit*2, меняем текстовое название месяца в

компоненте *Label*3. В компоненте *SpinEdit*2 можно установить ограничение на выбор максимального числа месяца 12. Результат этих действий показан на рис. 6.

Задание 3. В форму поместите компоненты *SpinEdit1, SpinEdit2, Label1, Label2, ComboBox1*. Свойству *Value* компонента *SpinEdit1* присвойте начальное значение 2012. Свойству *Value* компонента *SpinEdit2* присвойте начальное значение 1. У *Label1* подпишите год, у *Label2* – месяц. В *ComboBox1* перечислите названия месяцев года. Добавьте кнопку. Присвойте ей название «Дата». По щелчку по этой кнопке в компоненте *Label3* должно появиться название месяца, соответствующее значению переменной в компоненте *SpinEdit2*, и год, соответствующий значению в компоненте *SpinEdit1*. Изменяя значения в компонентах *SpinEdit1*, *SpinEdit2*, меняем текстовое название месяца в компоненте Label3. В компоненте *SpinEdit2* установите ограничение на выбор максимального числа месяца 12.

Пример 4

Метод *function ItemRect* (*Item: integer*) состоит в слудующием: *TRect* возвращает координаты прямоугольника, ограничивающего указанный в параметре *Item* элемент. В форме содержится список и четыре метки (*TLabel*). При создании формы в список включаются три строки. Когда пользователь выбирает одну из строк, тогда в компонентах *TLabel* отображаются координаты прямоугольника, который ограничивает выбранную строку. По кнопке *Add* добавим какие-нибудь значения в список. Отсортируем его (результат выполнения см. на рис. 7).

```
procedure TForm1.ListBox1Click(Sender: TObject);
var   ListBoxItem:TRect;
begin
    ListBoxItem:=ListBox1.ItemRect(ListBox1.ItemIndex);
    Label1.Caption:='Слева '+ IntToStr(ListBoxItem.Left);
```

```
    Label2.Caption:='Сверху '+ IntToStr(ListBoxItem.Top);
    Label3.Caption:='Справа '+ IntToStr(ListBoxItem.Right);
    Label4.Caption:='Снизу '+ IntToStr(ListBoxItem.Bottom);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    with ListBox1 do
    begin
        items.add('Привет') ;
        items.add('мы вместе') ;
        items.add('до свидания') ;
    end;
procedure TForm1.addClick(Sender: TObject);
begin
    ListBox1.Items.Add(Edit1.Text);
end;
procedure TForm1.SortClick(Sender: TObject);
begin
    ListBox1.Sorted:=True;
end;
```

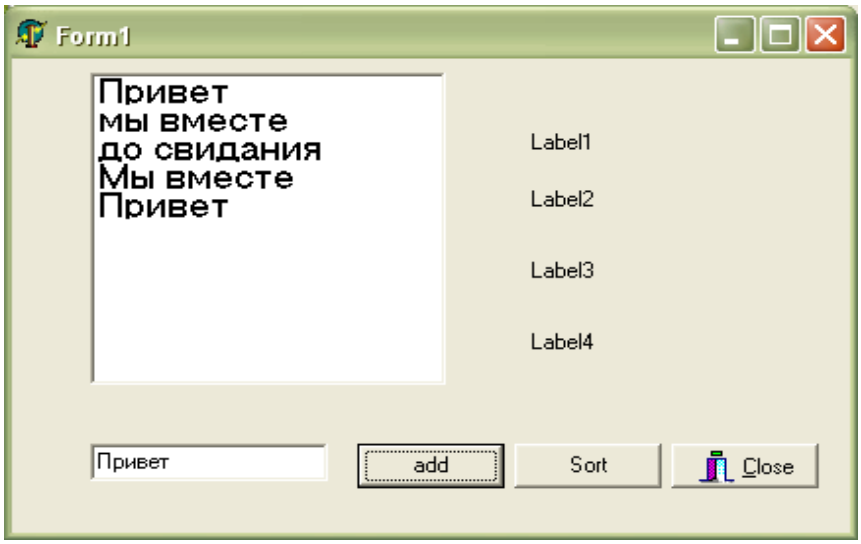

Рис. 7. Результат обработки события на кнопке *Sort*

Задание 4. Откройте новую форму. Поместите компоненты: *Edit1*, *Edit2*, *Edit3*, *Edit4*, *Button1*, *Button2*, *Button3*, *ListBox1*. Для кнопки *Button1* напишите процедуру закрытия формы. Для кнопки *Button1* напишите процедуру решения, например уравнения $Ax^2+By^2=C$. Уравнение задается пользователем в компоненте *Edit1*. Параметры *A, B, C* задаются в других компонентах *Edit* соответственно. Результат заносится в *ListBox1*. Для *Button3* напишите процедуру сортировки получившихся значений.

Пример 5

Свойство *ReadOnly* определяет, может ли пользователь изменить текст элементов управления. Следующая процедура изменит состояние *ReadOnly* поля ввода с *True* на *False* и, наоборот, с *False* на *True*, когда пользователь выполнит двойной щелчок на форме (результат выполнения показан на рис. 8).

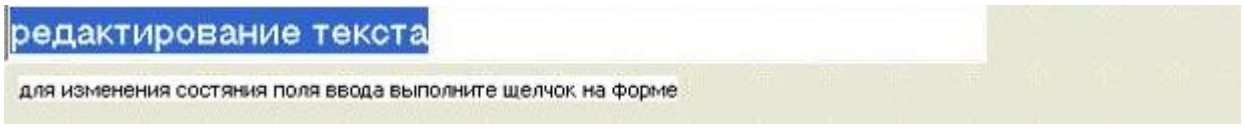

Рис. 8. Результат выполнения примера 5

```
procedure TForm1.FormActivate(Sender: TObject);
begin
    Edit1.Left:=2;
    Edit1.Top:=2;
    Edit1.ReadOnly:=true;
    Edit1.Text:='редактирование текста';
end;
procedure TForm1.FormDblClick(Sender: TObject);
begin
    Edit1.ReadOnly:= not Edit1.ReadOnly;
end;
procedure TForm1.FormPaint(Sender: TObject);
begin
```

Canvas.TextOut(10,40,'для изменения состояния поля ввода '+

'выполните щелчок на форме')

end;

Задание 5. Добавьте в форму компоненты *TMemo, TButton.* Напишите программный код записи всего текста или части текста из Edit1 в Memo1. Используйте методы *SelectAll, SelStart, SelLenth*. Добавьте в форму *Edit2, Edit3.* Укажите в новой строке поля *Memo1* количество знаков и с какого знака скопирована часть текста по выбору пользователя.

Пример 6

Компонент *TAnimate* предназначен для воспроизведения видеофильмов. В свойстве *CommonAVI* выберите любой *AVI*-клип. Для кнопки напишите следующую процедуру (результат показан на рис. 9):

procedure *TForm1.Button1Click(Sender: TObject);*

begin

Animate1.Play(Animate1.startFrame,Animate1.StopFrame,0);

end;

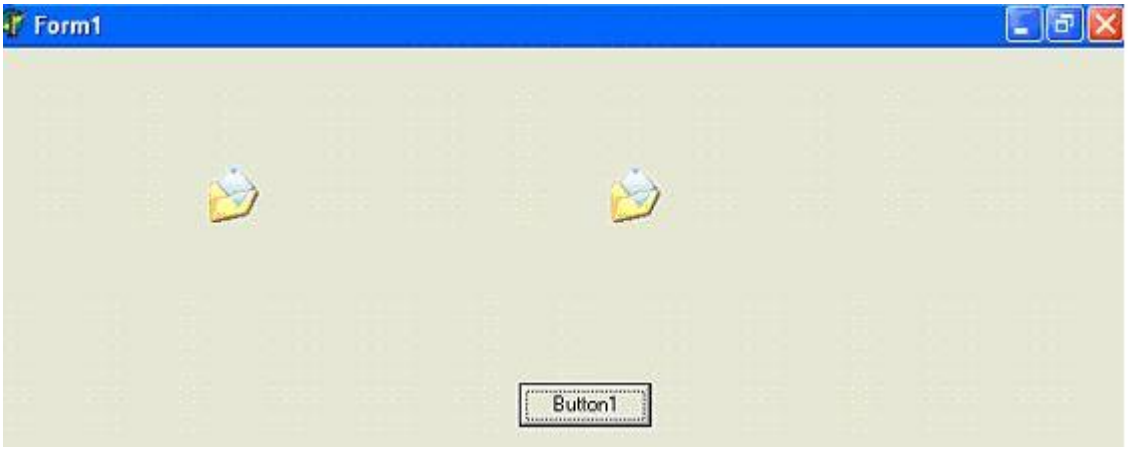

Рис. 9. Работа компонента *Animate*1

Задание 6. Добавьте необходимые компоненты (*Edit1, Edit2*) и отредактируйте программный код так, чтобы пользователь мог сам управлять

параметрами компонента *TAnimate*. Не забудьте всем кнопкам дать названия, понятные пользователю.

Пример 7

Форма содержит компонент *TDateTimePicker* и кнопку. Свойство *Kind* имеет значение *dtkTime*. Когда пользователь выполняет щелчок по кнопке, тогда появляется окно сообщения, в котором отображается текущее время (результат выполнения процедуры *Button1Click* показан на рис. 10).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    MessageDlg('Текущее время:'+ TimeToStr(DateTimePicker1.Time),
    mtInformation,[mbOk],0);
end;
```

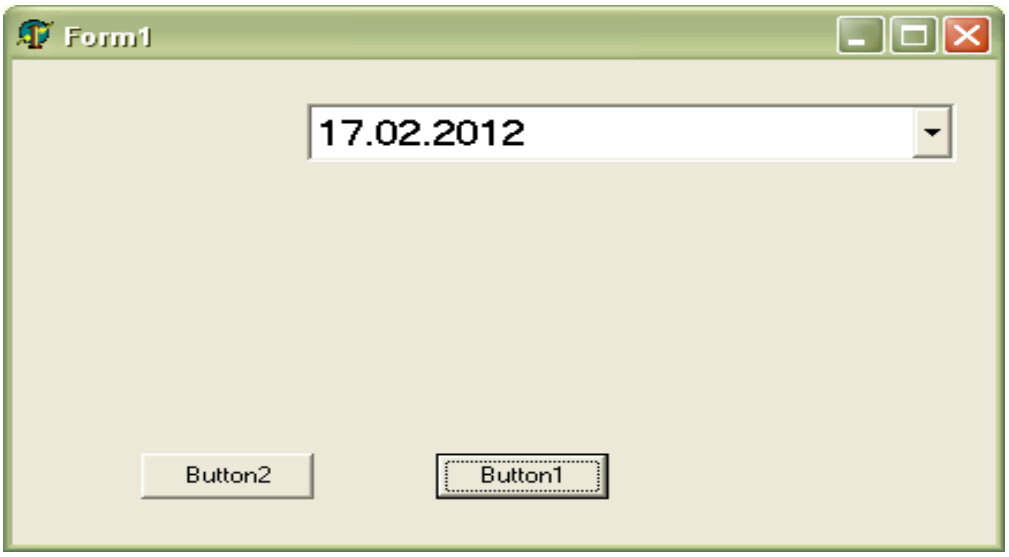

Рис. 10. Текущее время во время выполнения приложения

Можно поменять свойство *Kind* на *dtkDate* и установить текущую дату при открытии формы, используя функцию *Now*(). В компоненте списков заполнить список месяцев года. По кнопке в компоненте выделить месяц, выбранный на календаре. Для определения номера месяца можно использовать функцию *FormatDateTime*:

function *FormatDateTime(const Format: string; DateTime: TDateTime): string;*

Например: дата Пятница, 17 февраля 2012 года, время 10:30 утра .

S := FormatDateTime('"The meeting is on " dddd, mmmm d, yyyy, " at " hh:mm AM/PM', StrToDateTime('2/15/05 10:30am')).

Задание 7. Поменяйте свойство *Kind* на *dtkDate*. Установите текущую дату при открытии формы, используя функцию *Now*(). В компоненте списков заполните список месяцев года. По кнопке в компоненте выделите месяц, выбранный на календаре. Используйте функцию *FormatDateTime* для определения номера месяца.

Пример 8

В данном примере определена процедура *checkState*, которая с помощью *WinApi*-фукции *GetKeyState* проверяет состояние клавиши [*CapsLock*]. Если эта клавиша нажата, то в первом разделе строки состояния появляется надпись «ЗАГЛАВНЫЕ». Процедура *checkState* связана с обработчиком событий *FormKeyDown*, который вызывается при нажатии клавиши (рис. 11).

Рис. 11. Вид формы после включения клавиши *CapsLock*

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    checkState;
end;
procedure TForm1.checkState;
```

```
begin
  if  odd(GetKeyState(VK_CAPITAL)) then
     StatusBar1.Panels[0].Text:='ЗАГЛАВНЫЕ'
  else
     StatusBar1.Panels[0].Text:='';
end;
```

В разных панелях строки состояния можно показать текущую дату, время.

Задание 8. Покажите в разных панелях строки состояния текущую дату, время, режим замены/вставки, включение/выключение клавиши *Num Lock*.

Пример 9

Элемент управления *TShape* предоставляет пользователю набор геометрических фигур (примитивно), которые можно поместить в любом месте формы. Следующая процедура изменит форму, цвет и образец заполнения компонента (рис. 12).

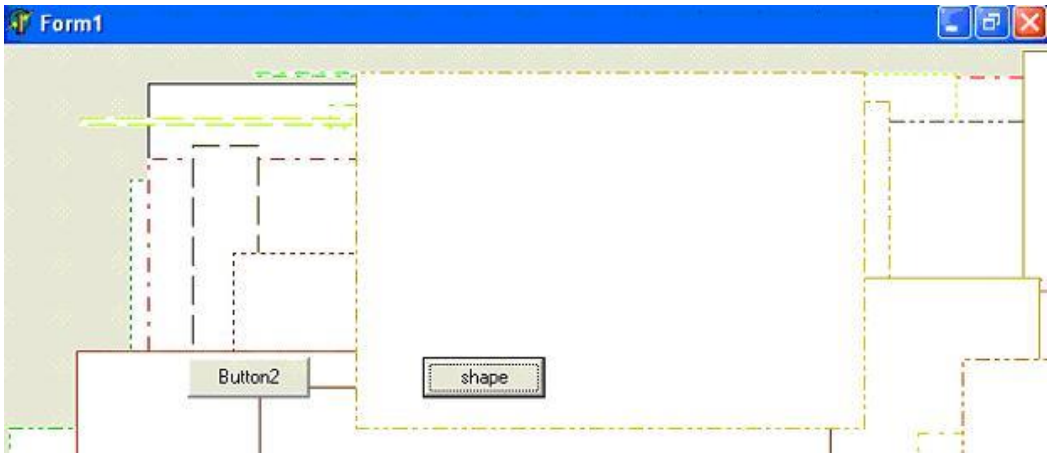

Рис. 12. Форма во время выполнения приложения

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     Shape1.Shape:=stEllipse;
     Shape1.Brush.Color:=clMaroon;
     Shape1.Brush.Style:=bsBDiagonal;
```

```
end;
```

Задание 9. Добавьте в форму еще один компонент *TShape*, *TTimer*. С помощью свойства *Pen* измените толщину прямоугольника. Добавьте следующий программный код. Запустите программу:

```
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

Пример 10

Рассмотрим работу с компонентом меню *TMenu*. Форма приложения содержит меню *MainMenu*1, компонент *Timer*1. Дизайнер меню активизируется двойным щелчком на соответствующем компоненте меню в форме, показанной на рис. 13. С помощью дизайнера меню создайте меню формы. Основные пункты: цвет (подпункты – синий, красный, восстановление цвета формы), управление пунктами меню (время, установка отметки), выход (результат выполнения пунктов меню представлен на рис. 14).

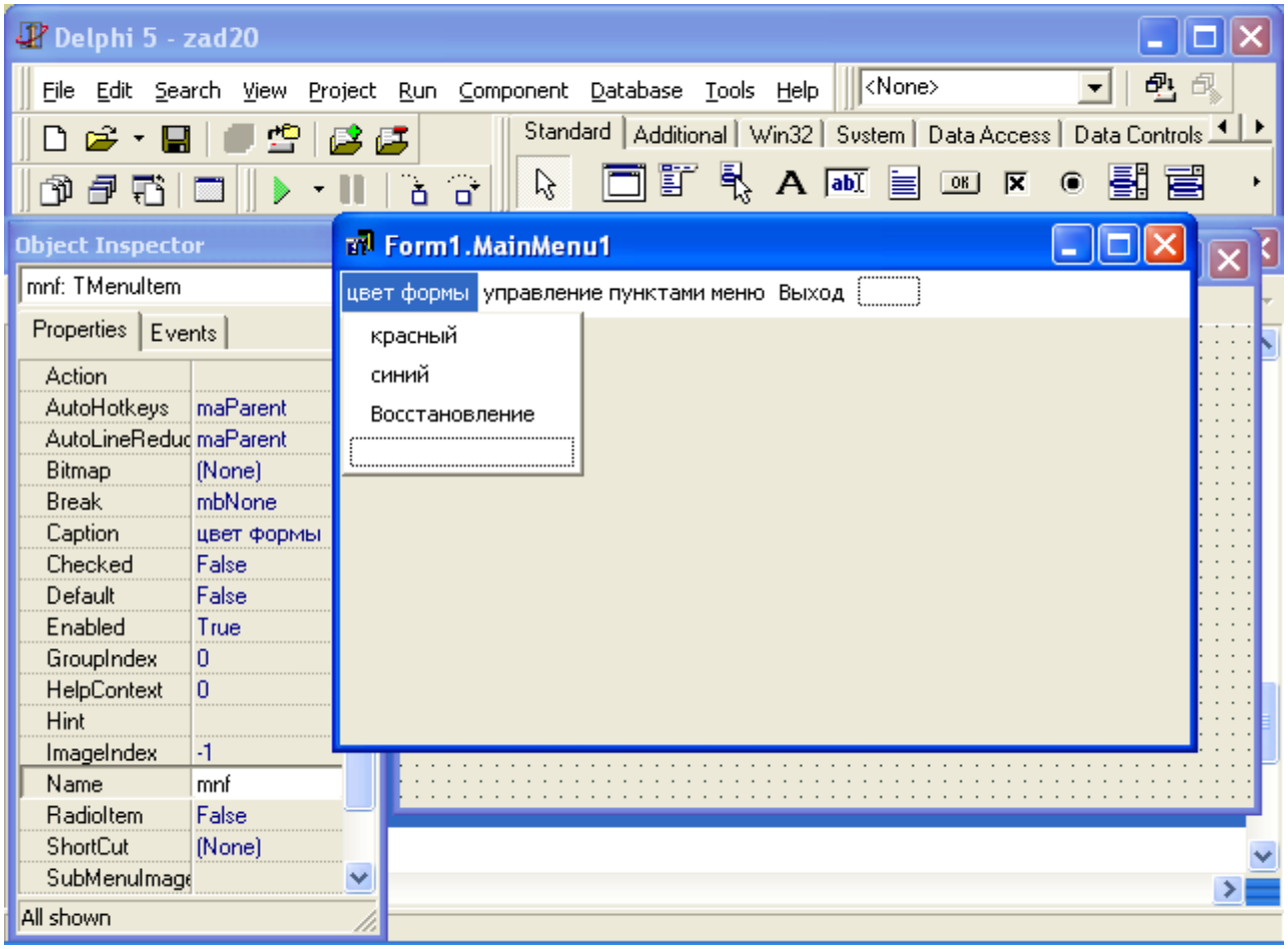

Рис. 13. Дизайнер меню

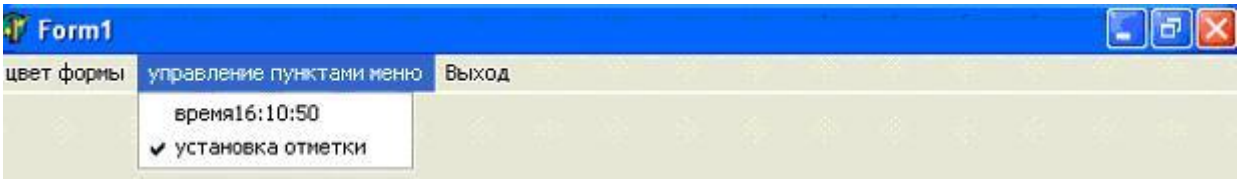

Рис. 14. Результат выполнения пунктов меню

```
unit Unit20;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ExtCtrls;
type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    mnf: TMenuItem;
    mnfred: TMenuItem;
    mnfbly: TMenuItem;
    mnfreset: TMenuItem;
    vt1: TMenuItem;
    mncl: TMenuItem;
    Nnmtime: TMenuItem;
    mnotmrtki: TMenuItem;
    Timer1: TTimer;
    procedure mnfredClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure mnfblyClick(Sender: TObject);
    procedure mnfresetClick(Sender: TObject);
    procedure mnclClick(Sender: TObject);
    procedure NnmtimeClick(Sender: TObject);
    procedure mnotmrtkiClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
```

```
  Form1: TForm1;
    FormColorMemory:LongInt;
implementation
{$R *.DFM}
procedure TForm1.mnfredClick(Sender: TObject);
begin
     Form1.Color:=clRed;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
FormColorMemory:=form1.Color;
end;
procedure TForm1.mnfblyClick(Sender: TObject);
begin
     Form1.Color:=clBlue;
end;
procedure TForm1.mnfresetClick(Sender: TObject);
begin
    form1.Color:=FormColorMemory;
end;
procedure TForm1.mnclClick(Sender: TObject);
begin
    close;
end;
procedure TForm1.NnmtimeClick(Sender: TObject);
begin
    Nnmtime.Caption:='Время'+TimeToStr(Time);
end;
procedure TForm1.mnotmrtkiClick(Sender: TObject);
begin
```

```
    if   mnotmrtki.Checked then
         mnotmrtki.Checked :=false
    else
    mnotmrtki.Checked :=true;
end;
end.
```

Задание 10. Поместите на форму компонент *TShape* в виде прямоугольника. В момент нажатия кнопки мыши на прямоугольник он исчезает. Когда кнопка будет отпущена, тогда прямоугольник снова появится. Используйте процедуры *Hide, Show,OnMouseUp, OnMouseDown*. Добавьте в описание класса формы поле типа *MouseBtn: Boolean*. В процедуре обработки события *OnMouseMove* определить цвет компонента *TShape*: свойству *Color* присваивается в зависимости от значения поля *MouseBtn* значение *clAgua* (голубой) или *clYellow* (желтый).

2. УПРАВЛЕНИЕ ПРОГРАММОЙ НА ОСНОВЕ СООБЩЕНИЙ О СОБЫТИЯХ В СРЕДЕ *DELPHI*

2.1. События и обработчики событий

С помощью *Delphi* можно создать приложения для *Windows*. Одним из свойств таких приложений является управление событиями. Программа выполняется на основе генерируемых сообщений о событиях, которые обрабатываются программным кодом приложения. Такой код необходимо написать для каждого события, на которое должна реагировать программа. Процедура, предназначенная для реагирования на какое-либо событие, в *Delphi* называется процедурой обработки события (*Event Handler*).

2.1.1. События

Выделяется две категории событий:

- события, обусловленные действиями пользователя;

- простые, или программно-управляемые.

Процедуры обработки пользовательских событий обеспечивают интерактивное взаимодействие приложений и пользователя. В *Delphi* для этой цели применяются предварительно определенные обработчики событий, которые могут использоваться практически всеми компонентами. Сюда относятся обработчики событий *OnClick, OnDbClick,* обработчики событий мыши и клавиатуры.

К обычным событиям относятся события активации, завершения, события изменения состояния компонентов и т. д., которые являются косвенным результатом действия пользователя.

Delphi генерирует процедуры обработки для каждого события и дает им имена в соответствии с именами компонентов, для которых эти процедуры предназначены. Например, в случае обработки события щелчка мышью (*Click*) на элементе управления *Button*1 генерируется пустая процедура обработки события, которая показана на рис. 15.

Рис. 15. Заголовок процедуры *OnClick* для кнопки *Button*1

Теперь необходимо вписать нужный программный код между зарезервированными словами *Object Pascal begin* и *end.*

2.1.2. Действия пользователя с мышью

Когда пользователь передвигает мышь, то по экрану перемещается небольшое изображение – указатель мыши. Различают следующие виды действий с мышью, на которые должно реагировать приложение:

- нажатие кнопки мыши (*MouseDown*);
- отпускание кнопки мыши (*MouseUp*);

- перемещение мыши (*MouseMove*);
- щелчок (*Click*);
- двойной щелчок (*DbClick*).

С каждым действием мыши связан определенный обработчик события. Эта информация представлена в табл. 1.

Таблица 1

Результаты действий пользователя с мышью

Действия пользователя	Состояние кнопки мыши	Обработчик событий мыши
Нажатие кнопки мыши	Кнопка нажата	*OnMouseDown*
Отпускание кнопки мыши	Кнопка не нажата	*OnMouseUp*
Перемещение мыши	Кнопка нажата или не отжата	*OnMouseMove*

2.1.3. События клавиатуры

События клавиатуры генерируются как только клавиша будет нажата или отпущена. В обоих случаях приложение получает сообщение от *Windows* о нажатии клавиши. *Windows* посылает в приложение сообщение, состоящее из двух частей: о нажатии клавиши и о символе. Это отражено в наличии двух обработчиков событий:

- *OnKeyDown* (вызывается при нажатии клавиши) и *OnKeyUp* (вызывается при отпускании клавиши). Данные события происходят при каждом нажатии клавиши;
- *OnKeyPress* – происходит лишь при нажатии клавиши, с которой связан читаемый символ.

2.1.4. Перехват событий клавиатуры

Для формы в *Delphi* определено свойство, которое может быть установлено таким образом, что события клавиатуры будут сначала проверяться и обрабатываться самой формой, после чего они могут быть

обработаны активным элементом управления. Это свойство *KeyPreview*. Если данное свойство имеет значение False, события клавиатуры поступают непосредственно к тому элементу управления, который активен для ввода. Если же этому свойству присвоено значение True, события клавиатуры сначала будет получат форма.

Объявление свойства *KeyPreview* выглядит следующим образом:

property KeyPreview: Boolean;

2.2. Примеры создания приложения с процедурами обработки события и задания к лабораторной работе № 2

Цель лабораторной работы № 2 – научиться управлять программой на основе сообщений о событиях в среде *Delphi*. Используя представленные в примерах процедуры обработки событий, выполнить задания.

Пример 1

Обработчики событий *OnMouseDown* и *OnMouseUp* имеют тип:

type *TMouseEvent = procedure (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer) of object;*

В процедуре обработки события следует определить, что должно произойти в качестве реакции на нажатие кнопки мыши. Используйте параметр *Button*, если для разных кнопок мыши должны быть определены разные действия. Компонент *TShape*, имеющий форму круга, создается и отображается при каждом нажатии кнопки мыши. Однако если нажимается правая кнопка, то появляющийся в данной форме круг имеет черный цвет и красный контур (рис. 16).

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
  var NewShape:TShape;
begin
   NewShape:=  TShape.Create(Form1);
```

```
    NewShape.Parent:=Self;
    NewShape.Shape:=stCircle;
    NewShape.Height:=13;
    NewShape.Width:=13;
    NewShape.Left:=X;
    NewShape.Top:=Y;
    NewShape.show;
    if   Button=mbRight then
    begin
      NewShape.Pen.Color:=clRed;
      NewShape.Brush.Color:=clBlack;
    end;
end;
```

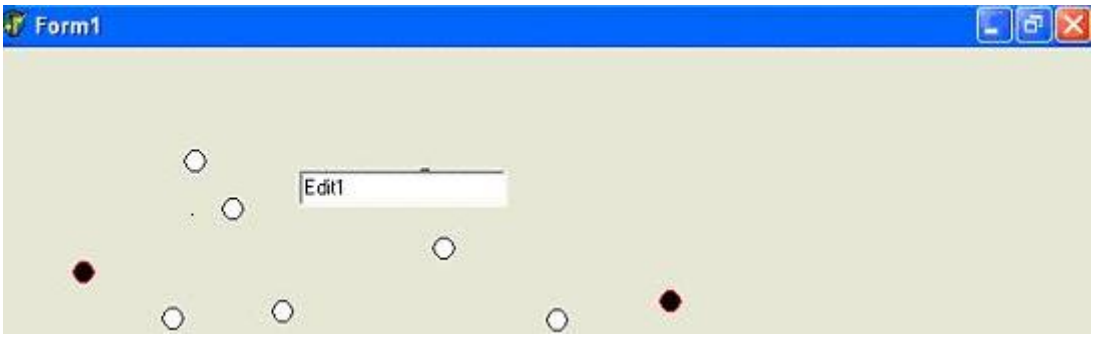

Рис. 16. Форма *Form*1 после обработки событий *OnMouseDoun*

Задание 1. Поместите на форму компонент *TShape* в виде прямоугольника. В момент нажатия кнопки мыши на прямоугольник он исчезает. Когда кнопка будет отпущена, тогда прямоугольник снова появится. Используя процедуры *Hide, Show,OnMouseUp, OnMouseDown*, добавьте в описание класса формы поле типа *MouseBtn: Boolean*. В процедуре обработки события *OnMouseMove* определите цвет компонента *TShape*: свойству *Color* присваивается в зависимости от значения поля *MouseBtn* значение *clAgua* (голубой) или *clYellow* (желтый).

Пример 2

В данном примере используется форма *Form*1, содержащая кнопку и группирующую рамку. После выполнения щелчка по кнопке она перемещается в группирующую рамку (рис. 17). Это обеспечивается путем задания свойству *Parent* кнопки значения *GroupBox*1. Двойной щелчок на *GroupBox*1 выводит *Button*1 за пределы группирующей рамки.

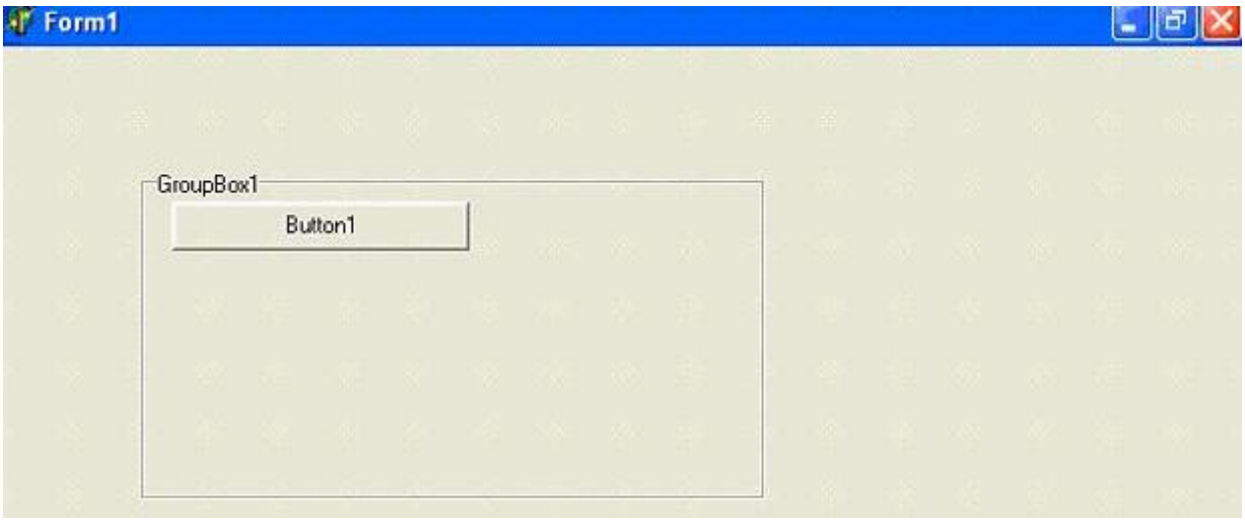

Рис. 17. Вид формы после нажатия на кнопку *Button*1

Ниже приведены соответствующие процедуры обработки событий:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.Parent:=GroupBox1;
end;
procedure TForm1.GroupBox1DblClick(Sender: TObject);
begin
    Button1.Parent:=Form1;
end;
```

Задание 2. Добавьте в форму компонент *Memo*1. Покажите в нем имя родительского компонента. Используя компонент *DialogOpen*, вставьте в компонент *Memo* любой текстовый файл (лучше небольшой по размеру). Добавьте в форму кнопки *Button*2, *Button*3, *Button*4. Напишите процедуры

выравнивания текста по правому, левому краю и центрирования соответственно.

Пример 3

В форме находится панель (*Panel*1), опция (*CheckBox*1), поле ввода (*Edit*1) и кнопка. Свойство *DragMode* имеет значение *DmAutomatic*. Когда пользователь выполняет щелчок на опции *CheckBox*1, тогда начинается процесс ее перетаскивания. Когда перемещение компонента завершается, значение параметра *Targe*t в методе *CheckBox1EndDrag* определяет, какой текст появляется в окне сообщения. В поле ввода *Edit*1 отображается имя компонента, на котором была оставлена опция (форма после выполнения этих действий выглядит так, как показано на рис. 18).

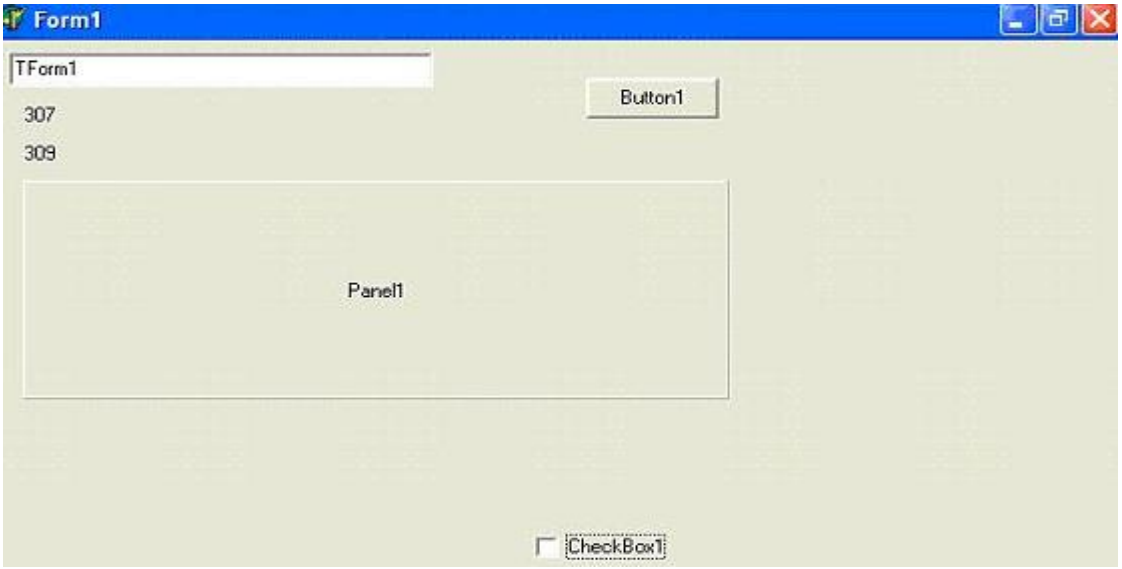

Рис. 18. Положение компонента *ChechBox*1 после перемещения на форму

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
    Accept:=true;
end;
procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
    Accept:=true;
end;
procedure TForm1.CheckBox1EndDrag(Sender, Target: TObject; X, Y:
Integer);
begin
    if Target<> nil  then
        Edit1.Text:= Target.ClassName
    else
        Edit1.Text:='нельзя перетащить объект в данном компоненте';
    if Target= Panel1  then
    begin
        showMessage('Оставлена на панели');
        CheckBox1.Left:=X+Panel1.Left;
        CheckBox1.Top:=Y+Panel1.top;
    end
    else begin
        if Target=Form1 then
        showMessage('Оставлена на Form1 ');
        CheckBox1.Left:=X;
        CheckBox1.Top:=Y;
    end;
        Label1.Caption:= IntTostr(X);
        Label2.Caption:= IntTostr(Y);
    CheckBox1.Left:=X;
    CheckBox1.Top:=Y;
end;
```

Задание 3. Пусть в форме содержится три опции. Свойство *DragMode* каждой из них имеет значение *dmAutomatic*. Используя функцию *Dragging*,

определите, перетаскивается объект или нет. Когда опция перетаскивается, измените цвет формы на *clAgua*, *clYellow*, *clLime* соответствующий каждой опции.

Пример 4

События клавиатуры выполняются процедурой обработки события того компонента, который в данный момент активен для ввода. Зададим клавишу, например *F*6, с помощью которой пользователь может увеличить изображение на экране. Для компонента *Image*1 зададим его свойство *Picture* и поместим его в центр. Нажмем клавишу *F*6 в компоненте *Edit*1 (см. рис. 19).

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if Key=VK_f6 then
     Edit1.Text:='F6 нажата';
    Image1.stretch:=true;
end;
```

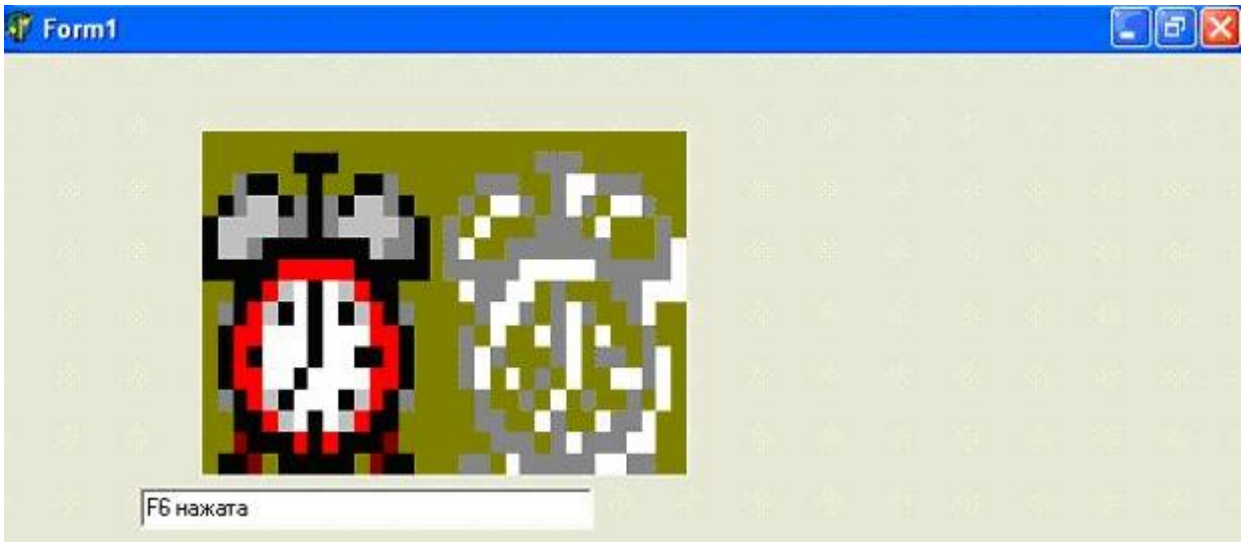

Рис. 19. Форма после нажатия клавиши *F*6

Задание 4. Установите свойство формы *KeyPreview – true*. Определите, что изменилось. Выведите предупреждение пользователю, что цифровая

клавиатура включена/не включена. Предупреждение выведите в компоненте *StatusBar*1 в *Panel*1, а сообщение из примера в *Panel*2.

Пример 5

Пример демонстрирует последовательность, в которой события клавиатуры обрабатываются приложением. Когда программа запускается в первый раз, тогда при нажатии клавиш [*Ctrl+Alt*] форма приобретает фиолетовый цвет. Если выполнить щелчок по кнопке *Button*1, то эта кнопка перестанет быть активной (см. рис. 20). При щелчке по кнопке *Button*2 свойство *KeyPreview* формы приобретет значение *True*. Если теперь нажаты клавиши в сочетании [*Ctrl+Alt*], то форма станет голубой, т. к. как она обрабатывает события нажатия клавиш:

```
procedure TForm1.Button1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if (Shift=([ssAlt, ssCtrl])) then
    form1.Color:=clPurple;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.Enabled:=false;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    form1.KeyPreview:=True;
end;
 procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if (Shift=([ssAlt, ssCtrl])) then
```

```
    form1.Color:=clAqua;
end;
```

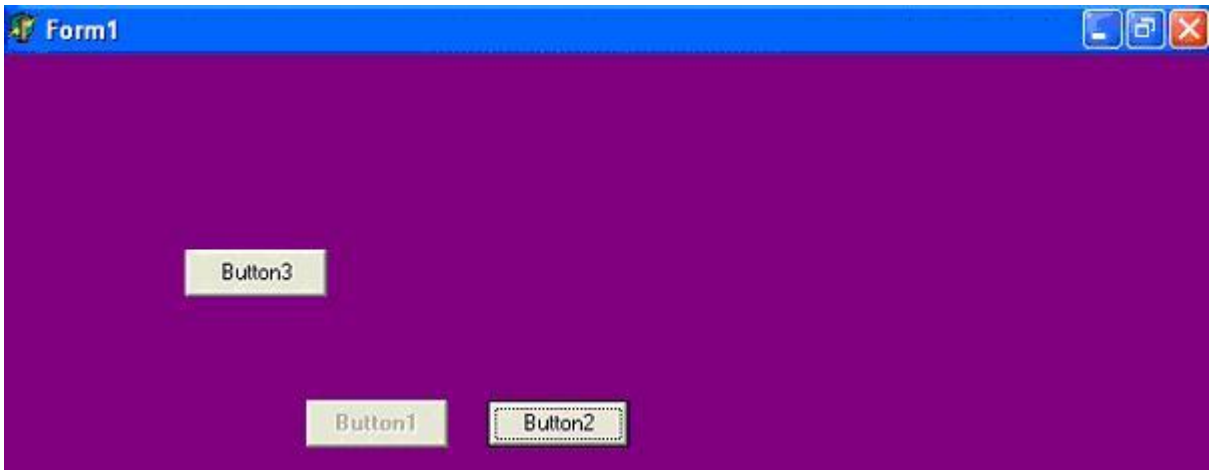

Рис. 20. Результат нажатия сочетания клавиш [*Ctrl+Alt*]

Задание 5. Поменяйте элемент управления табулятором и нажмите сочетание клавиш [*Ctrl+Alt*]. Запустите эту форму еще раз. Добавьте еще одну кнопку *Button*3. Напишите обработчик события: кнопка *Button*3 активна, цвет кнопок *Button*1 и *Button*2 при сочетании клавиш [*Ctrl+F*6] меняется произвольным образом. Используйте функцию *Random*.

Пример 6

При нажатии клавиши обычно генерируется целый ряд событий. В примере используется список, отображающий, в какой последовательности обрабатываются нажатия клавиш с помощью обработчика событий *OnkeyDown, OnKeyUp* и *OnKeyPress*. Функция *ShortCutToText* объявлена в модуле *Menus*. Поскольку в данном разделе форма не имеет меню, постольку этот модуль не включается автоматически в разделе *Uses* модуля, поэтому вы должны добавить туда имя этого модуля самостоятельно. Поле *Edit*1 должно быть активным для ввода (см. рис. 21):

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    ListBox1.Items.Add('Edit1.KeyDown-'+ShortCutToText(Key));
```

```
end;
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    ListBox1.Items.Add('Edit1.KeyPress-'+Key);
end;
procedure TForm1.Edit1KeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    ListBox1.Items.Add('Edit1.KeyUp-'+ShortCutToText(Key));
end;
```

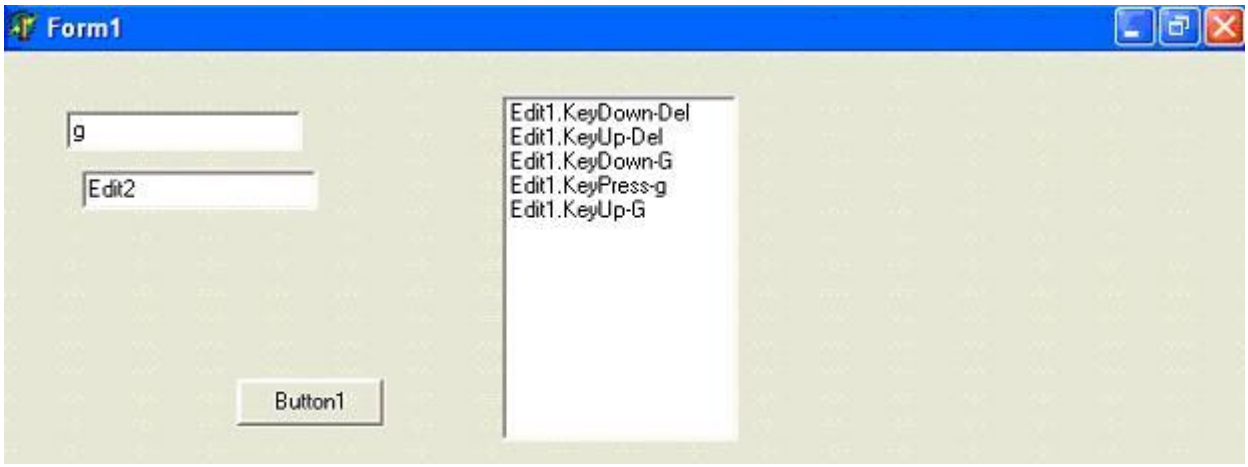

Рис. 21. Результат вызовов обработчиков событий *OnKeyDown*, *OnKeyPress*, *OnKeyUp*

Задание 6. Измените параметр функции *ShortCutToText(ShortCut: TShortCut): string* так, чтобы можно было увидеть сочетание клавиш. Используя функцию *TextToShortCut(Text: string): TshortCut*, покажите в каком-нибудь компоненте сочетание нажимаемых клавиш на компоненте *Edit*2.

Пример 7

Следующие процедуры обработки событий *FormKeyDown* и *FormKeyUp* включают и отключают изображения в компоненте *Image* и изменяют текст в компоненте панели *Panel*1, когда клавиша *Z* нажимается (см. рис. 22) и затем отпускается.

В этом примере параметр *Key* преобразуется в тип *Char*.

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if chr(Key)='Z'  Then
    begin
        Image1.Visible:=true;
        panel2.Caption:= 'Загрузка судна в порту';
    end;
end;
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    if chr(Key)='Z'  Then
    begin
        Image1.Visible:=false;
        panel2.Caption:='';
    end;
end;
```

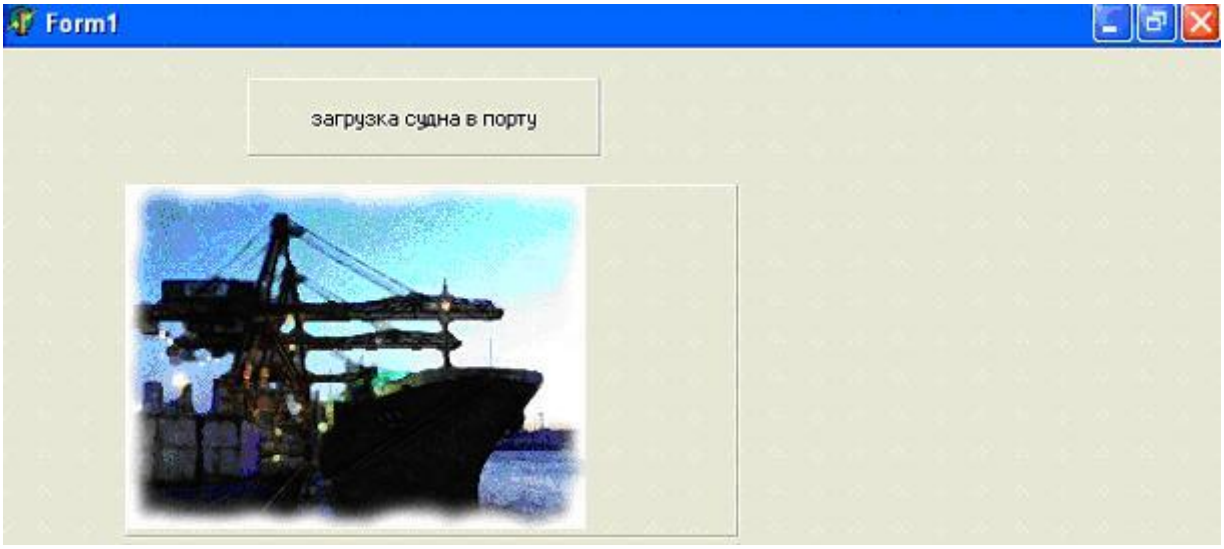

Рис. 22. Появление изображения и текста при отжатой клавише

Задание 7. При нажатии клавиши *Ins* в компоненте панели *Panel*1 должен отобразиться текст. При нажатии сочетания клавиш [*Alt+F*10] появится сообщение: «нажаты *Alt+F*10». Используйте функцию:

function *MessageDlg(const Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx: Longint): Word.*

Пример 8

Форма содержит панель, меню и компонент *Bevel*. В свойствах *Hint* каждого компонента напишите справочный текст (в каждом пункте меню тоже). Процедура *OnHint* является обработчиком события компонента *Tapplication*, поэтому написана процедура обработки *OnHint*. Новый метод *DisplayHintPanel* включен в форму в процедуре *OnCreate* главной формы (результат работы программы показан на рис. 23).

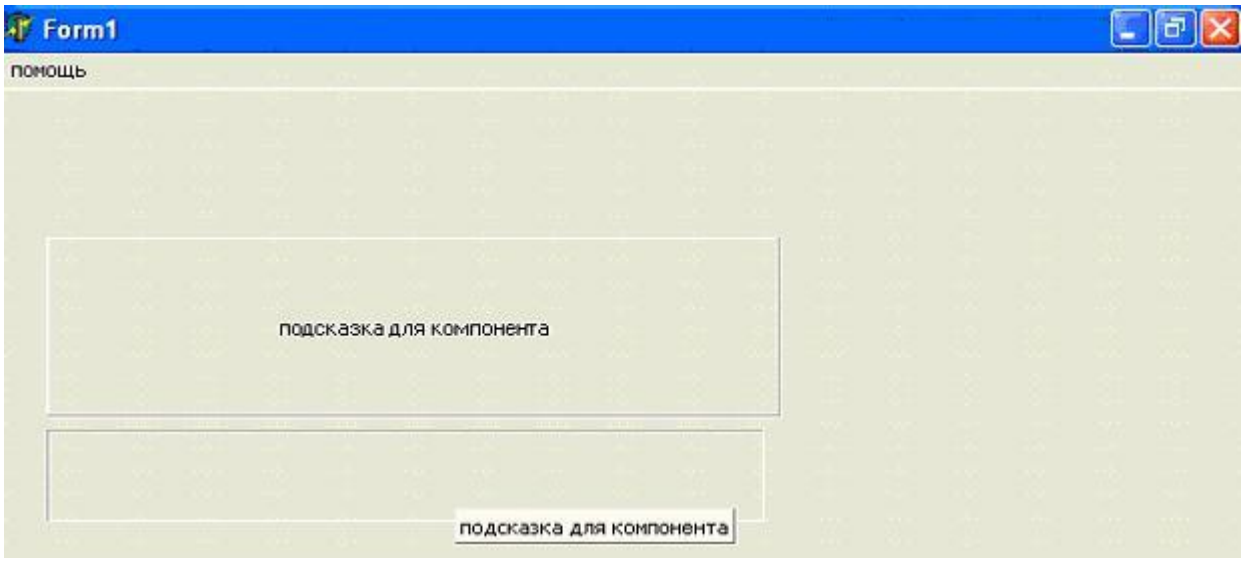

Рис. 23. Отображение подсказок для компонентов

```
unit Unit17;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ExtCtrls;
type
```

```
  TForm1 = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    Bevel1: TBevel;
    N2: TMenuItem;
    N1: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
  procedure DisplayHintPanel(Sender: TObject);
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
     Application.OnHint:=DisplayHintPanel;
end;
procedure TForm1.DisplayHintPanel(Sender: TObject);
begin
     Panel1.caption:= Application.hint;
end;
end.
```

Задание 8. В форму добавьте кнопку. По нажатии клавиши добавьте новый пункт меню. Подсказка содержит текст «подсказка к пункту» и название пункта. Текст названия пункта меню можно передать из компонента *Edit*.

Пример 9

В форме помещены компоненты: кнопка, *OpenPictureDialog1* и *Image*1. После открытия окна диалога в компонент *Image* вставляется выбранный файл с рисунком (пример работы программ показан на рис. 24).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenPictureDialog1.Execute;
  Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
```

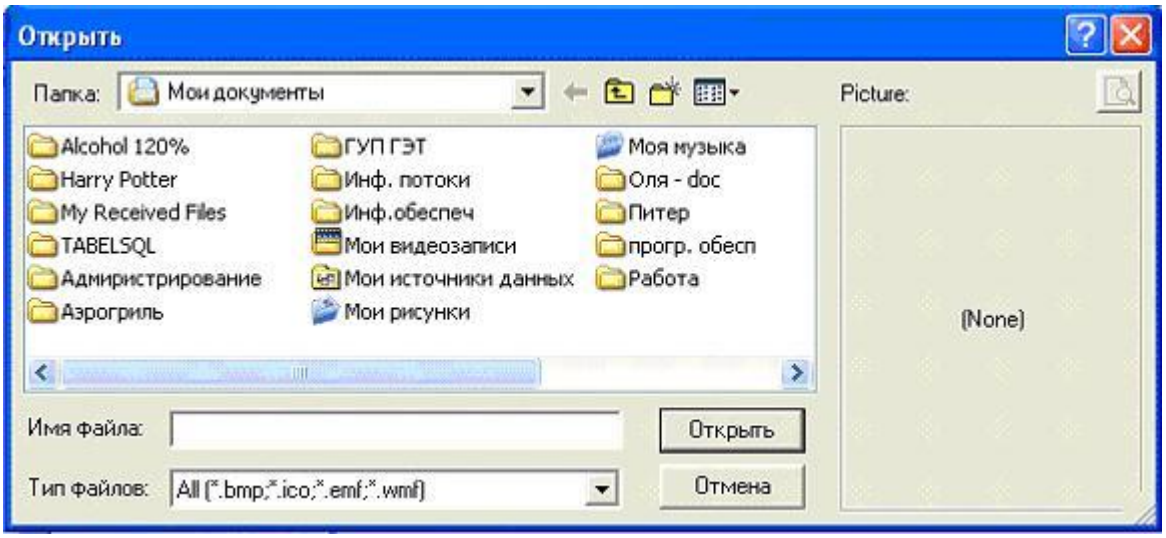

Рис. 24. Открытие компонента *OpenPictureDialog*1 для поиска рисунка

Задание 9. Настройте компонент *OpenPictureDialog*1 для открытия только рисунков с расширением **.bmp*. Настройте свойства *Image*1 так, чтобы рисунок занимал всю форму и входил полностью в компонент.

Пример 10

Для работы с документами используются приложения: *MDI* (*Multiple Document Interface*) – интерфейс для одновременной работы со многими

документами и *SDI* (*Single Document Interface*) – интерфейс для работы с одним документом. В *MDI*-приложениях два или более окон могут быть активными. В *SDI*-приложениях это невозможно. Различные документы имеют общее рабочее пространство, называемое родительским окном. В *MDI*-приложении родительское окно всегда является главной формой приложения. В нем возможны другие окна, в которых могут быть открыты различные документы. Эти окна не могут размещаться за пределами окна приложения. Они называются дочерними окнами.

В проект поместите четыре формы. В форме *Unit*1 в свойстве *FormStyle* установите значение *fsMDIForm* – родительскую форму. Все остальные формы определите как дочерние с помощью установки значения *fsMDIChild*. Свойству *Visible* всех форм должно быть задано значение *True*. Включите в форму компонент *TMainMenu*. Назовите его *Window*. В это меню добавьте пункты *Cascade*, *Tile* и т.д. С их помощью можно управлять окнами *MDI*-приложения (рис. 25).

Рис. 25. Форма управления окнами

```
procedure TForm1.Tile1Click(Sender: TObject);
begin
    TileMode:=tbVertical;
    Tile;
end;
procedure TForm1.ArrangeIcons1Click(Sender: TObject);
begin
    ArrangeIcons;
end;
procedure TForm1.Cascade1Click(Sender: TObject);
```

```
begin
    Cascade;
end;
procedure TForm1.Next1Click(Sender: TObject);
begin
    Next;
end;
procedure TForm1.Previous1Click(Sender: TObject);
begin
    Previous;
end;
procedure TForm1.CLOSE1Click(Sender: TObject);
begin
    if ActiveMDIChild <> nil then
    activeMDIChild.close;
end;
end.
```

Задание 10. Каждой форме присвойте свой заголовок, используя свойство *Caption*. Для каждой дочерней формы установите разные значения свойства *WindowState*. На каждой дочерней форме расположите по одной кнопке *Button*1. Для каждой кнопки напишите процедуру изменения свойств другой формы (*WindowState, Position, BorderStyle, Font*).

Глоссарий для среды *Delphi*

Active – свойство типа *Boolean* – определяет активность формы, приложения, открытие или закрытие набора данных.

Acr – рисование дуги.

Add – метод, обеспечивающий добавление строк в список.

Align – свойство типа *Talign* – определяет вариант выравнивания компонента внутри контейнера.

Application – объект типа *TApplication* с именем *Application*, которое автоматически создается при каждом запуске приложения *Delphi*.

Clear – метод, служащий для очистки содержимого компонентов (текстовой информации).

Close – метод закрытия формы.

ComponentCount – число принадлежащих компонентов.

ComponentIndex – номер компонента в списке принадлежащих компонентов.

ComponentState – состояние текущего компонента.

Create – метод создания экземпляров формы.

Delete – метод удаления строк из списка.

Ellipse – рисование заполненного эллипса.

Execute – функция, используемая для вызова любого стандартного диалога. При закрытии диалога с помощью кнопки *ОК* (*Открыть* или *Сохранить*) функция *Execute* возвращает значение *True*, а при отмене диалога – значение *False*.

FillRect – заполнение прямоугольной области.

Form – компонент класса *Tform*; на основе формы начинается конструирование приложения.

FormStyle – свойство типа *TFormStyle* для определения стиля формы.

FrameRect – вывод незаполненного прямоугольника.

Free – метод уничтожения формы.

Hide – метод управления видимостью. Процедура *Hide* скрывает форму, устанавливая ее свойству *Visible* значение *False*.

Hint – свойство типа *String* – задает текст подсказки, отображаемый в том случае, когда курсор находится в области компонента и некоторое время неподвижен.

Insert – метод, задающий позицию в списке, на которую вставляется элемент.

InputBox – функция, которая отображает диалоговое окно, содержащее для ввода строку текста.

Items – свойство типа *Tstring* – представляет собой массив строк и определят количество элементов списка и их содержимое.

KeyPreview – свойство типа *Boolean* – определяет, будет ли форма обрабатывать события клавиатуры прежде, чем их обрабатывают элементы управления формы.

LineTo – рисование линии от указателя до точки с координатами *X* и *Y*.

MaxLength – свойство, определяющее, какое максимальное число символов можно ввести в поле ввода, элемент управления *Memo* или поле со списком.

Message – свойство типа *String* – содержит описание исключительной ситуации.

MessageDlg – функция, отображающая окно сообщений в центре экрана. Она позволяет получить ответ пользователя. Параметр *Msg* содержит выводимое сообщение.

MessageDlgPos – функция, отличающаяся от функции *MessageDlg* наличием параметров *X* и *Y*, управляющих положением окна на экране.

SelLength – свойство, возвращающее длину отмеченной в полк ввода подстроки (в символах).

SelectAll – метод, выделяющий в элементе весь текст.

SelStart – свойство, устанавливающее начало отмеченного диапазона.

Show – процедура, отображающая форму в немодальном режиме, при этом свойству *Visible* присваивается значение *True*. Сама форма переводится на передний план.

ShowMessage – процедура, отражающая окно сообщения с кнопкой *OK*. Заголовок содержит название исполняемого файла приложения, а строка *Msg* выводится как текст сообщения.

Name – имя компонента.

OnClick – событие нажатия, возникающее при выборе управляющего элемента.

OnKeyDown – событие, которое непрерывно генерируется при нажатии клавиши и удержании ее нажатой.

OnKeyPress – событие, которое генерируется при нажатии клавиши.

OnKeyUp – событие, которое происходит только после отпускания клавиши.

Owner – владелец компонента.

Parent – свойство типа *TwinControl* – указывает на родительский элемент управления для компонента.

Pie – вывод фигуры в форме сектора круга.

Polygon – рисование заполненного многоугольника.

PlyLine – рисование незаполненного многоугольника.

ReadOnly – свойство, которое определяет, может ли пользователь изменить текст элементов управления.

Rectagle – вывод заполненного прямоугольника.

Refresh – метод, используемый для обновления элементов управления, состоящего в удалении изображения элемента и его перерисовке.

RoundRect – вывод заполненного прямоугольника со скругленными краями.

Run – команда, которая компилирует, компонует и затем запускает на выполнение созданный *EXE*-файл приложения.

TabOrder – свойство типа *TtabOrder* – определяет порядок получения компонентами контейнера фокуса при нажатии клавиши <*Tab*>, т. е. последовательность обхода (табуляции) компонентов.

Tag – целое число, хранимое вместе с компонентом.

Text – свойство типа *Tcaption*; содержит строку, связанную с компонентом. Это содержимое компонента.

Value – свойство типа *Variant* – это физические данные в объекте.

Var – ключевое слово для объявления раздела переменных.

3. СОЗДАНИЕ БАЗЫ ДАННЫХ В СУБД *MYSQL*

3.1. Основы баз данных

Информационные системы позволяют автоматизировать сбор и обработку данных. Они являются банками данных, включающими в себя:

- вычислительную систему;
- базу данных (БД);
- систему управления базами данных (СУБД);
- набор прикладных программ.

БД обеспечивает хранение информации и представляет собой совокупность данных, организованных по определенным правилам. БД позволяет структурировать, хранить и обрабатывать данные различного типа.

СУБД – это совокупность языковых и программных средств, предназначенных для создания, ведения и использования БД. Персональная СУБД обеспечивает возможность создания локальных СУБД. К ним относятся *Paradox, DBase, FoxPro, Access*. Многопользовательские СУБД позволяют создавать информационные системы, функционирующие в архитектуре клиент-сервер. К ним относятся *Oracle1*[1], *InterBase*[2], *Microsoft SQL Server*[3], *MySQL*[4].

Разработку и сопровождение *MySQL*, самой популярной *SQL*-базы данных с открытым кодом, осуществляет компания *MySQL AB*. *MySQL AB* – коммерческая компания, основанная разработчиками *MySQL*, строящая свой бизнес, предоставляя различные сервисы для СУБД *MySQL*.

Прикладные программы, или приложения, служат для обработки данных, содержащихся в БД. Пользователь осуществляет управление БД и работу с ее данными именно с помощью приложения.

---

[1] *http://www.oracle.com/index.html*

[2] *http://www.embarcadero.com/*

[3] *http://www.microsoft.com/sqlserver/ru/ru/*

[4] *http://www.mysql.com/*

3.2. Таблицы базы данных

MySQL – это система управления реляционными базами данных. В реляционной базе данные хранятся не все вместе, а в отдельных таблицах, благодаря чему увеличивается скорость и гибкость. Таблицы связываются между собой при помощи отношений, которые при выполнении запроса объединяют данные из нескольких таблиц. *SQL* как часть системы *MySQL* является языком структурированных запросов, включает в себя наиболее распространенный стандартный язык, используемый для доступа к базам данных.

MySQL поддерживает несколько типов полей, которые показаны в табл. 2.

Таблица 2

Типы данных полей *MySQL*

Тип	Название	Описание обозначения
1	2	3
TINYINT	Очень малое целое число	Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255
SMALLINT	Малое целое число	Диапазон со знаком от -32 768 до 32 767. Диапазон без знака от 0 до 65 535
MEDIUMINT	Целое число среднего размера	Диапазон со знаком от -8 388 608 до 8 388 607. Диапазон без знака от 0 до 16 777 215
INT *INTEGER*	Целое число нормального размера	Диапазон со знаком от -2 147 483 648 до 2 147 483 647. Диапазон без знака от 0 до 4 294 967 295

Продолжение табл. 2

1	2	3
BIGINT	Большое целое число	Диапазон со знаком от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807. Диапазон без знака от 0 до 18 446 744 073 709 551 615
FLOAT(точность)	Число с плавающей точкой	Атрибут точности может иметь значение меньше либо равно 24 для числа с плавающей точкой обычной (одинарной) точности
DOUBLE[(*M*,*D*)]	Число с плавающей точкой удвоенной точности нормального размера	Допустимые значения: от -1,7 976 931 348 623 157 E+308 до -2,2 250 738 585 072 014 E-308, 0, и от 2,2 250 738 585 072 014 E-308 до 1,7 976 931 348 623 157 E+308
REAL[(*M*,*D*)]	Число с плавающей точкой удвоенной точности нормального размера	Данные обозначения являются синонимами для *DOUBLE*
DECIMAL[(*M*[,*D*])]	«Неупакованное» число с плавающей точкой	Ведет себя подобно столбцу *CHAR*
NUMERIC[(*M*[,*D*])]	«Неупакованное» число с плавающей точкой	Данные обозначения являются синонимами для *DECIMAL*
DATE	Дата	Поддерживается интервал от «1000-01-01» до «9999-12-31»

Окончание табл. 2

1	2	3
DATETIME	Комбинация даты и времени	Поддерживается интервал от «1000-01-01 00:00:00» до «9999-12-31 23:59:59»
TIMESTAMP[(*M*)]	Временная метка	Интервал от «1970-01-01 00:00:00» до некоторого значения времени в 2037 году
YEAR[(2|4)]	Год в двухзначном или четырехзначном форматах	*MySQL* выводит значения *YEAR* в формате *YYYY*
TIME	Время	Интервал от «-838:59:59» до «838:59:59». *MySQL* выводит значения *TIME* в формате «*HH:MM:SS*»
CHAR(*M*)	Строка фиксированной длины	Диапазон аргумента *M* составляет от 0 до 255 символов
VARCHAR(*M*)	Строка переменной длины	Концевые пробелы удаляются при сохранении значения

Предупреждение: следует помнить, что при выполнении вычитания между числовыми величинами, одна из которых относится к типу *UNSIGNED*, результат будет беззнаковым!

3.3. Работа с пакетом *MySQL-Front*

Пакет *MySQL-Front* предназначен для работы с таблицами базы данных *MySQL*. С помощью *MySQL-Front* можно:

- создавать и удалять базы *MySQL*, таблицы;
- создавать, редактировать и удалять поля таблиц;
- добавлять, редактировать и удалять записи в таблицах;

- редактировать и выполнять *SQL*-запросы к таблицам.

При открытии *MySQL-Front* появляется окно диалога, в котором надо создать контакт с сервером (рис. 26). Прописать параметры: название контакта *Description*, например «*localhost*»; название или *IP*-адрес сервера *Hostname* – *localhost*; имя пользователя *User* – *root*.

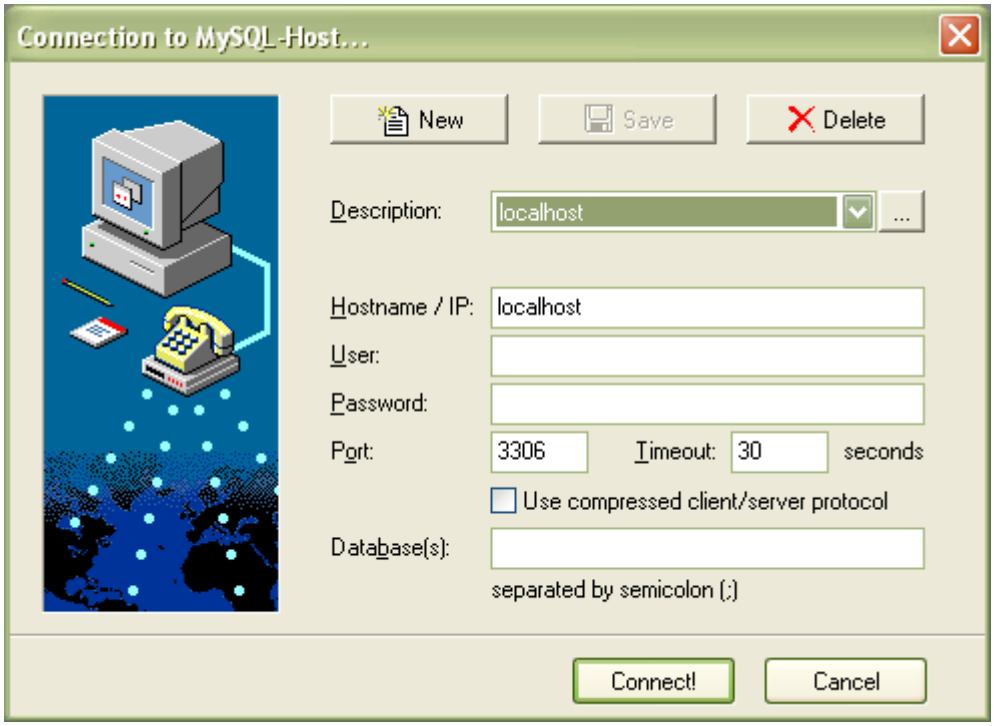

Рис. 26. Создание контакта с сервером

После нажатия кнопки *Connect*! произойдет соединение с сервером (см. рис. 27). Откроется окно программы *MySQL-Front*. В левой части окна из древовидной структуры выберите нужную базу данных. Выберите базу данных *test*. База данных стала доступной для просмотра и редактирования в правой части окна.

3.4. Создание таблиц базы данных

В базе *test* откройте окно *Database*. В левом окне появится список таблиц, входящих в эту базу данных. По пустому пространству окна щелкните правой клавишей мыши. Выберите строчку «*Create new Table*…». Появится окно диалога создания таблицы (см. рис. 28). Заполните необходимые поля: название таблицы, поля таблицы, типы полей.

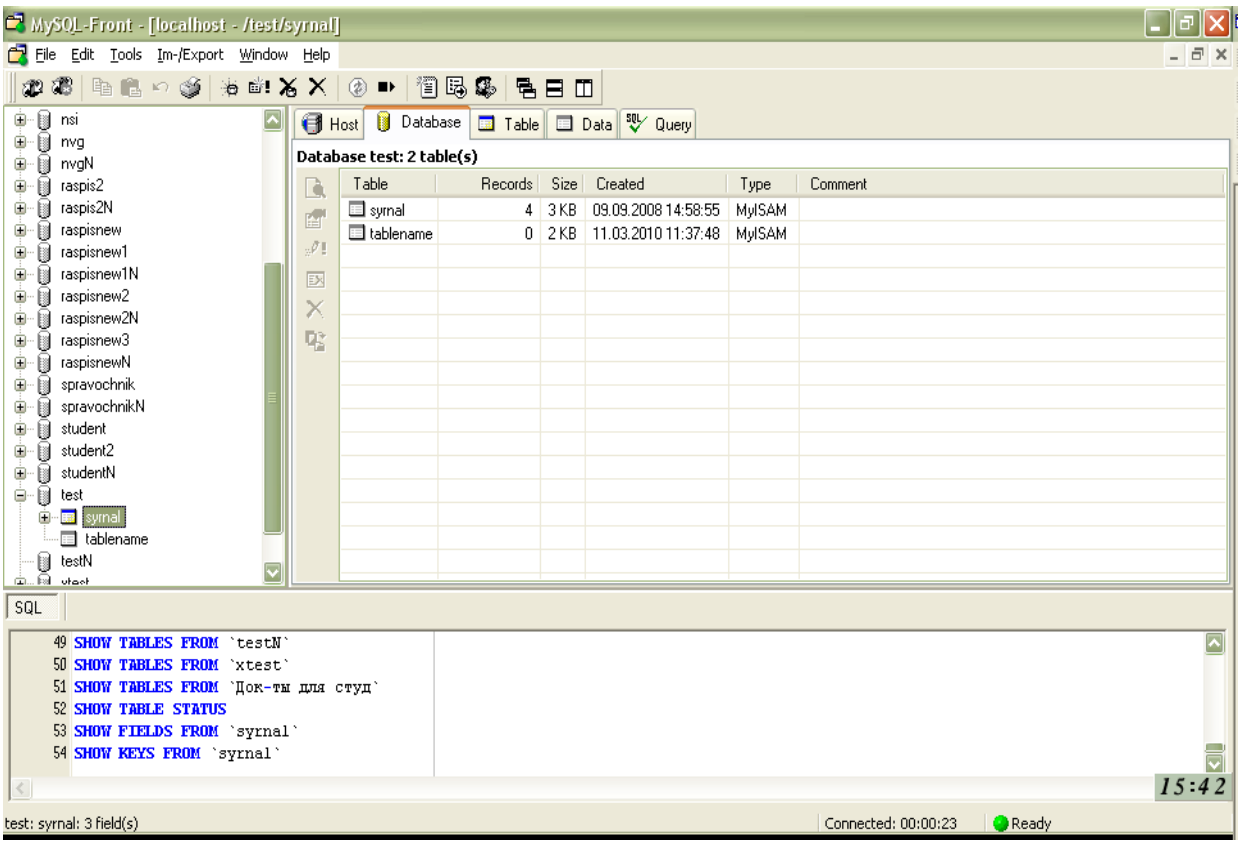

Рис. 27. Окно программы *MySQL-Front*

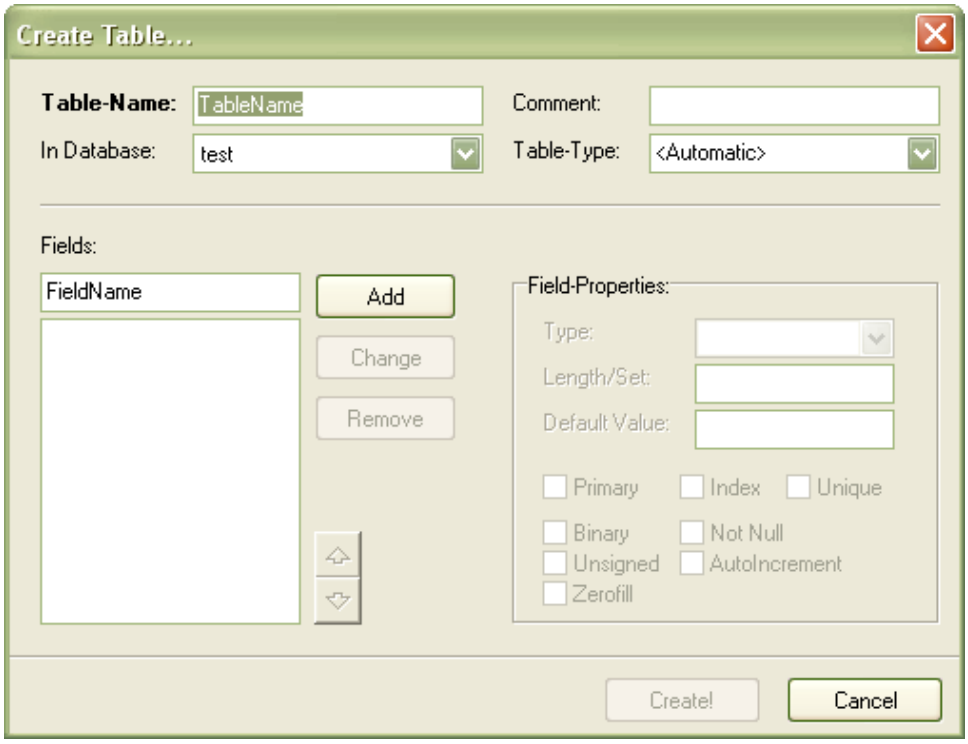

Рис. 28. Окно диалога *Create Table…*

Чтобы назначить индексы или ключи таблицы, надо по свободному месту правого окна щелкнуть правой клавишей мыши. Выберите строчку «*Add Field*

or Index…». Появится окно диалога создания нового поля. Выберите вкладку *Indexes* (рис. 29). Щелкните по кнопке *Add Primary*. Назначьте поля для ключа из списка *Available Columns* и стрелкой переместите их в колонку *Columns used*. Сохраните ключ, нажав на кнопку *Update Keys*.

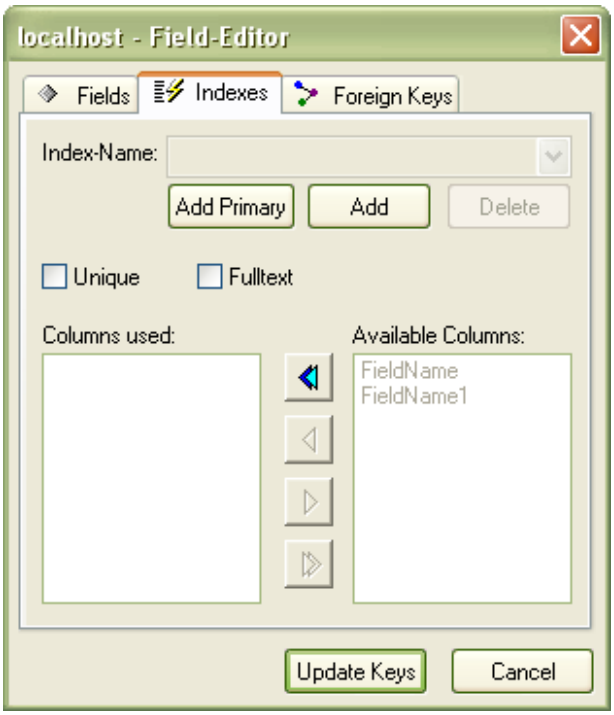

Рис. 29. Окно назначения индекса таблицы

Файлы таблиц имеют следующие расширения:

- *MYD* – файл данных;
- *MYI* – индексный файл (главный индекс (ключ) и вторичные индексы);
- *frm* – файл определения таблицы (формы) (параметры для проверки данных и целостности ссылок).

Созданные таблицы сохраняются в папке *C:\xampp\mysql\data\test*.

В начале выполнения лабораторной работы проверьте включение базы данных *MySQL* (см. рис. 30). Для этого на рабочем столе найдите иконку *XAMPP Control Panel*. Напротив раздела *MySQL* должно быть на зеленом фоне написано слово Running. Если напротив раздела *MySQL* ничего не написано, то нажмите на кнопку *Start*. В белом окне внизу появится текст: MySQL service started – и появится надпись на зеленом фоне: Running. Сервер запущен. Можно работать с таблицами базы данных *MySQL*.

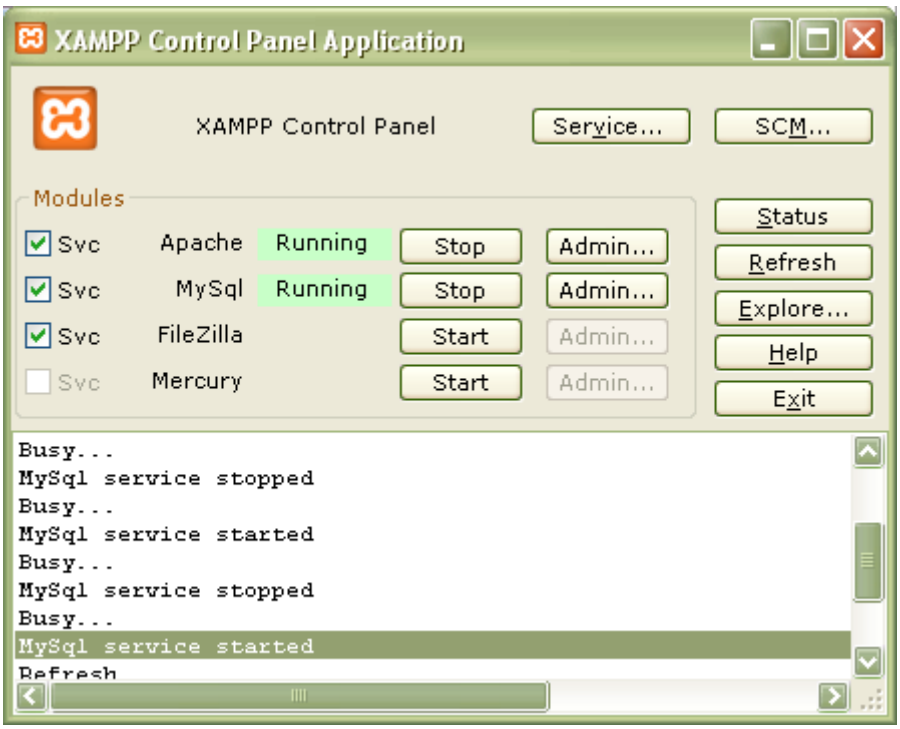

Рис. 30. Панель контроля работы сервера *MySQL*

Задания к лабораторной работе № 3

Цель лабораторной работы № 3 – научиться устанавливать СУБД *MySQL*, познакомиться с прикладным приложением *MySQL-Front* и научиться создавать таблицы в приложении *MySQL-Front*.

1. Установите дистрибутивы:

- запустите дистрибутив пакета *XAMPP*. Необходимо согласиться со всеми предложенными параметрами;
- автоматически установится имя пользователя *root*. Пароль – пустое значение;
- запустите дистрибутив пакета *MySQL-Front*. Необходимо согласиться со всеми предложенными параметрами;
- установите драйвер источника данных *MySQL*. Нужно согласиться со всеми предложенными параметрами.

2. Ознакомьтесь с заданием (см. в приложении задание 1). Вариант следует уточнить у преподавателя.

3. Нарисуйте *ER*-диаграммы, построенные в три этапа, по выданному заданию.
4. Разработайте таблицы в первой, второй и третьей нормальной форме.
5. Создайте таблицы в базе данных *test* на сервере *MySQL*.

4. СИСТЕМА ДОСТУПА К БД *MYSQL* И СОЗДАНИЕ ПРИЛОЖЕНИЙ ТИПА КЛИЕНТ/СЕРВЕР В СРЕДЕ *DELPHI*

4.1. Средства для работы с базами данных

К средствам *Delphi*, предназначенным для работы с БД, относятся следующие:

- инструментальные средства (специальные программы и пакеты, обеспечивающие обслуживание БД вне разрабатываемых приложений);
- компоненты, предназначенные для создания приложений, которые осуществляют операции с БД.

Для операций с БД система *Delphi* предлагает следующие инструментальные средства:

- *Borland Database Engine* (*BDE*) – процессор баз данных, который представляет собой набор библиотек, предназначенных для организации доступа к БД из приложений *Delphi*;
- *BDE Administrator* – утилита для настройки *BDE*. Позволяет настраивать различные параметры БД;
- *Database Desktop* – программа для создания и редактирования таблиц, *SQL*- и *QBE*-запросов;
- *SQL Explorer* – проводник БД, позволяющий настраивать параметры БД.

Компоненты, связанные с БД, делятся на визуальные и невизуальные:

- невизуальные компоненты – для организации доступа к данным, содержащимся в таблицах. Они представляют собой промежуточное звено между данными таблиц БД и визуальными компонентами;

- визуальные компоненты – для создания интерфейсной части приложения. С их помощью пользователь может выполнять такие операции с таблицами, как просмотр или редактирование данных.

На странице *BDE* в окне *Tool Palette* (рис. 31) находятся невизуальные компоненты, предназначенные для управления данными с использованием *BDE*:

- *Table* – набор данных, основанных на таблице БД;
- *Query* – набор данных, основанных на *SQL*-запросе;
- *StoredProc* – набор данных, основанных на процедуре, которая хранится на сервере;
- *DataBase* – соединение с БД;
- *Session* – текущий сеанс работы с БД;
- *BatchMove* – выполнение операций над группой записей;
- *UpdateSQL* – модификация набора данных, основанного на *SQL*-запросе;
- *NestedTable* – вложенная таблица.

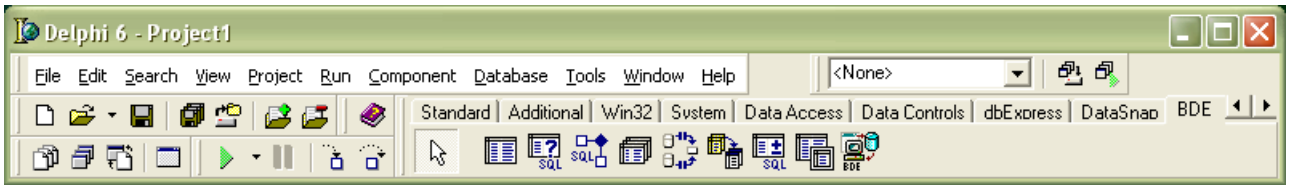

Рис. 31. Страница *BDE* в *Delphi6*

На странице *Date Access* в окне *Tool Palette* (рис. 32) находятся невизуальные компоненты, с помощью которых можно организовать доступ к данным:

- *Data Source* – источник данных;
- *ClientDataSet* – клиентский набор данных;
- *DataSetProvaider* – провайдер набор данных.

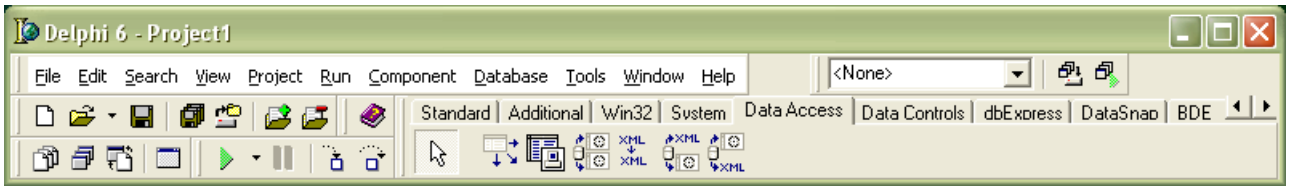

Рис. 32. Страница *Data Access* в *Delphi6*

На странице *Data Controls* (рис. 33) находятся визуальные компоненты, предназначенные для управления данными:

- *TDBGrid* – сетка (таблица);
- *TDBNavigator* – навигационный интерфейс;
- *TDBText* – надпись;
- *TDBEdit* – однострочный редактор;
- *TDBMemo* – многострочный редактор;
- *TDBImage* – графический образ;
- *TDBListBox* – простой список;
- *TDBComboBox* – комбинированный список;
- *TDBCheckBox* – независимый переключатель;
- *TDBRadioGroup* – группа независимых переключателей;
- *TDBLookupListBox* – простой список, формируемый по полю другого набора данных;
- *TDBLookupComboBox* – комбинированный список, формируемый по полю другого набора данных;
- *TDBRichEdit* – полнофункциональный текстовый редактор;
- *TDBCtrlGrid* – модифицированная сетка.

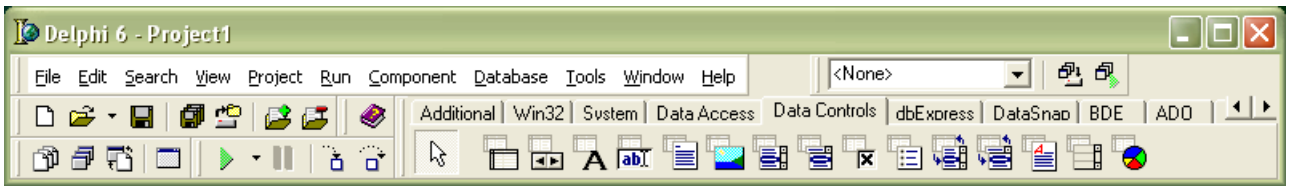

Рис. 33. Страница *Data Controls* в *Delphi6*

4.2. Технология создания формы приложения

В качестве примера использования возможностей *Delphi* для работы с БД рассмотрим технологию создания простого приложения. Основные этапы создания простого приложения:

- создание таблиц БД (см. п. 3.4);

- формы приложения.

В качестве примера рассмотрим форму приложения, с помощью которого можно перемещаться по записям таблицы БД, просматривать и редактировать поля записей, вставлять в таблицу новые записи, а также удалять из таблицы записи.

Вид формы приложения представлен на рис. 34. На форме расположены следующие компоненты: *Table1*, *DataSource1*, *DBGrid1* и *DBNavigator*1.

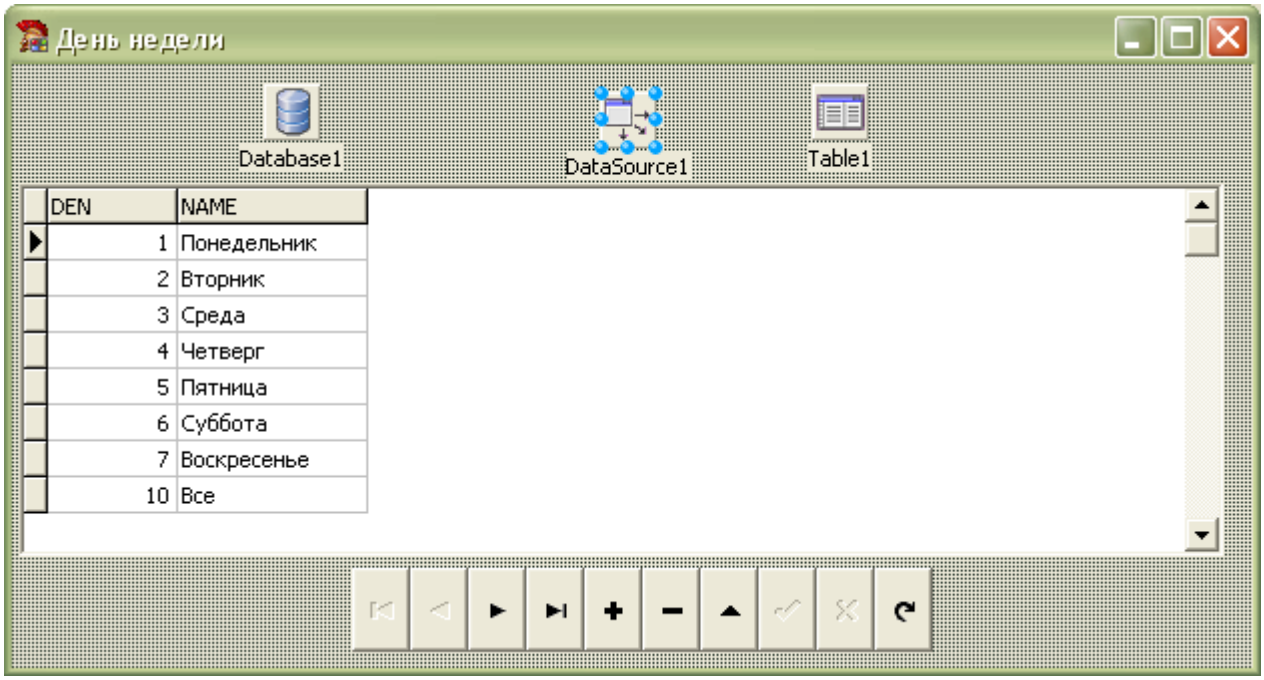

Рис. 34. Форма приложения для работы с БД

Компонент *Table*1 обеспечивает взаимодействия с БД. Для связи с требуемой таблицей необходимо установить соответствующие значения свойств *DataBaseName*, которое указывает путь к БД, и *TableName*, которое задает имя таблицы (см. рис. 35). После задания таблицы БД свойству *Active* должно быть установлено значение *True*.

При смене значений свойств *DataBaseName* или *TableName* нужно установить значение *False* свойству *Active*.

Имя таблицы лучше выбирать из раскрывающегося списка в поле значения свойства *TableName*. Если путь к БД (свойство *DataBaseName*) задан правильно, то в этом списке отображаются все доступные файлы.

Рис. 35. Свойства компонента *Table*1

Компонент *DataSourse*1 является промежуточным звеном между компонентом *Table*1, который соединен с реальной таблицей БД, и управляющими компонентами *DBGrid*1 и *DBNavigator*1, с помощью которых пользователь взаимодействует с этой таблицей (см. рис. 36). На компонент *Table*1, с которым связан компонент *DataSourse*1, указывает свойство *DataSet* последнего.

Компонент *DataBase*1 используется для соединения с БД (см. рис. 37). Свойство *AliasName* типа *String* указывает псевдоним БД. На этапе разработки приложения псевдоним выбирается из списка объектов в Интернете. Свойство

DatabaseName типа *String* задает имя БД, используемое только в приложении для организации подключения к БД.

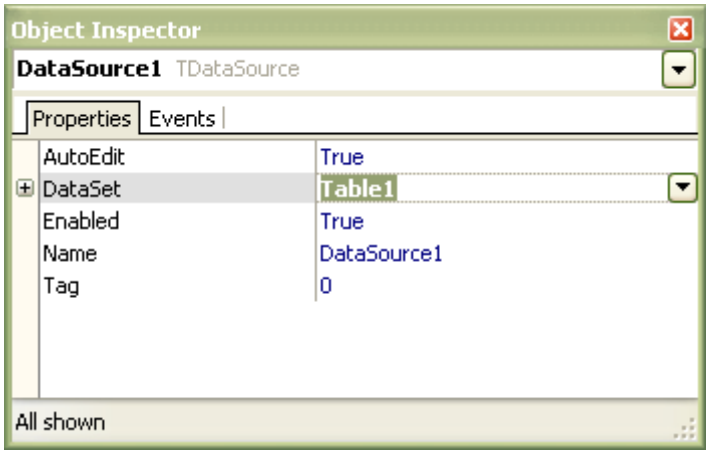

Рис. 36. Свойства компонента *DataSourse*1

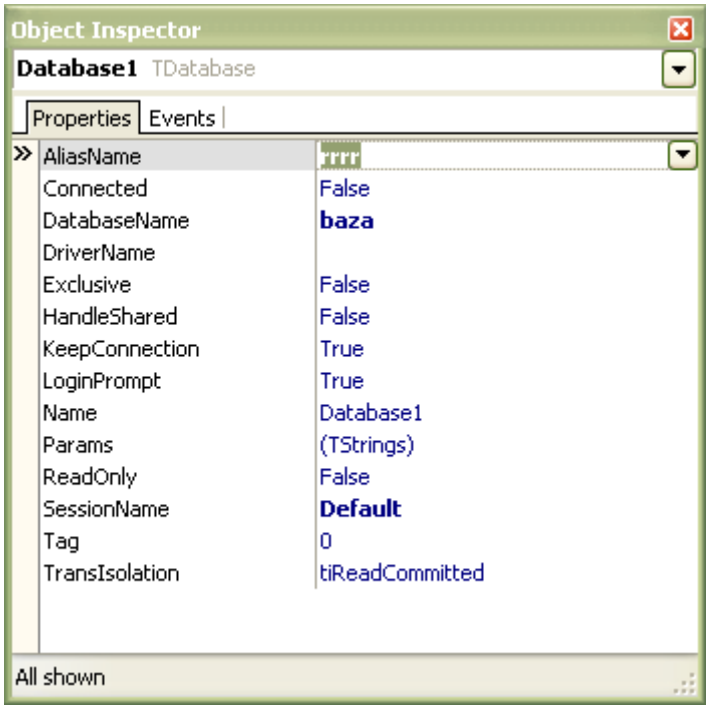

Рис. 37. Свойства компонента *Database*1

Компонент *DBGrid*1 отображает содержимое таблицы БД в виде сетки, в которой столбцы соответствуют полям, а строки – записям. По умолчанию пользователь может просматривать и редактировать данные. Компонент *DBNavigator*1 предоставляет возможность перемещаться по таблице, редактировать, вставлять и удалять записи. Компоненты *DBGrid*1 и

*DBNavigator*1 связываются со своим источником данных – компонентом *DataSourse*1 – через свои свойства *DataSource*.

При разработке приложения значения всех свойств компонентов можно задать с помощью инспектора объектов. Требуемые значения выбираются из раскрывающихся списков. В табл. 3 приведены компоненты для работы с БД, а также основные свойства и их значения. В данном примере используется таблица *den* из базы данных *test*. Путь к местоположению базы данных *test* указывает псевдоним *rrrr* из списка возможных *Alias* баз данных, прописанных на данном компьютере.

Таблица 3

Значения свойств компонентов

Компонент	Свойства	Значения
*Table*1	*Active* *DatabaseName* *TableName*	*True* (таблица *den* открыта) *baza* *den*
*DataBase*1	*AliasName* *Connected* *DatabaseName*	*rrrr* (псевдоним базы данных) *True* *baza*
*DataSourse*1	*DataSet*	*Table1*
*DBGrid*1	*DataSource*	*DataSource1*
*DBNavigator*1	*DataSource*	*DataSource1*

4.3. Программа *BDE Administrator*

Программа *BDE Administrator* – администратор процессора баз данных *BDE*. Для вызова администратора *BDE* запускается файл *bdadmin.exe*. Администратор позволяет настраивать параметры БД и системные параметры (операционной системы). Параметры псевдонима: название, тип, путь. Параметры драйвера: тип, язык. Системные установки: установки по умолчанию, форматы даты, времени и числовые параметры.

Для настройки некоторого параметра в левой части окна администратора *BDE* выбирается нужный объект, после чего в правой части окна становится доступным список параметров этого объекта. Добавить новый объект можно, выбрав в окне администратора пункт меню *Object/New* (*Объект/Новый*).

4.4. Работа с псевдонимами

Псевдоним (*alias*) указывает на местоположение БД и представляет собой специальное имя для обозначения каталога.

Для добавления нового псевдонима перед вызовом пункта меню *Object/New* администратора нужно выбрать вкладку *Database* в левой части окна. В диалоговом окне нужно выбрать тип драйвера. Для работы с базой данный *MySQL* нужно выбрать пункт *MySQL* (рис. 38).

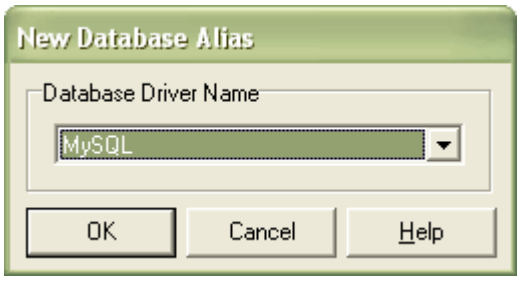

Рис. 38. Выбор типа драйвера

После нажатия кнопки *ОК* создается новый псевдоним, и его данные отображаются в окне администратора *BDE* (см. рис. 39). Новый псевдоним автоматически получает имя *ODBC*1 и параметры по умолчанию. Переименуем псевдоним в название *test*.

Псевдоним для работы с базой данных *MySQL* имеет следующие параметры:

- *TYPE* – указывает тип базы данных *MySQL*;
- *DATABASE NAME* – указывает имя удаленной базы данных;
- *ODBC DNS* – указывает источник данных;
- *SQLQRYMODE* – задает режим выполнения запросов (локальный или сетевой вариант);

- *LANGDRAVER* – определяет драйвер языка.

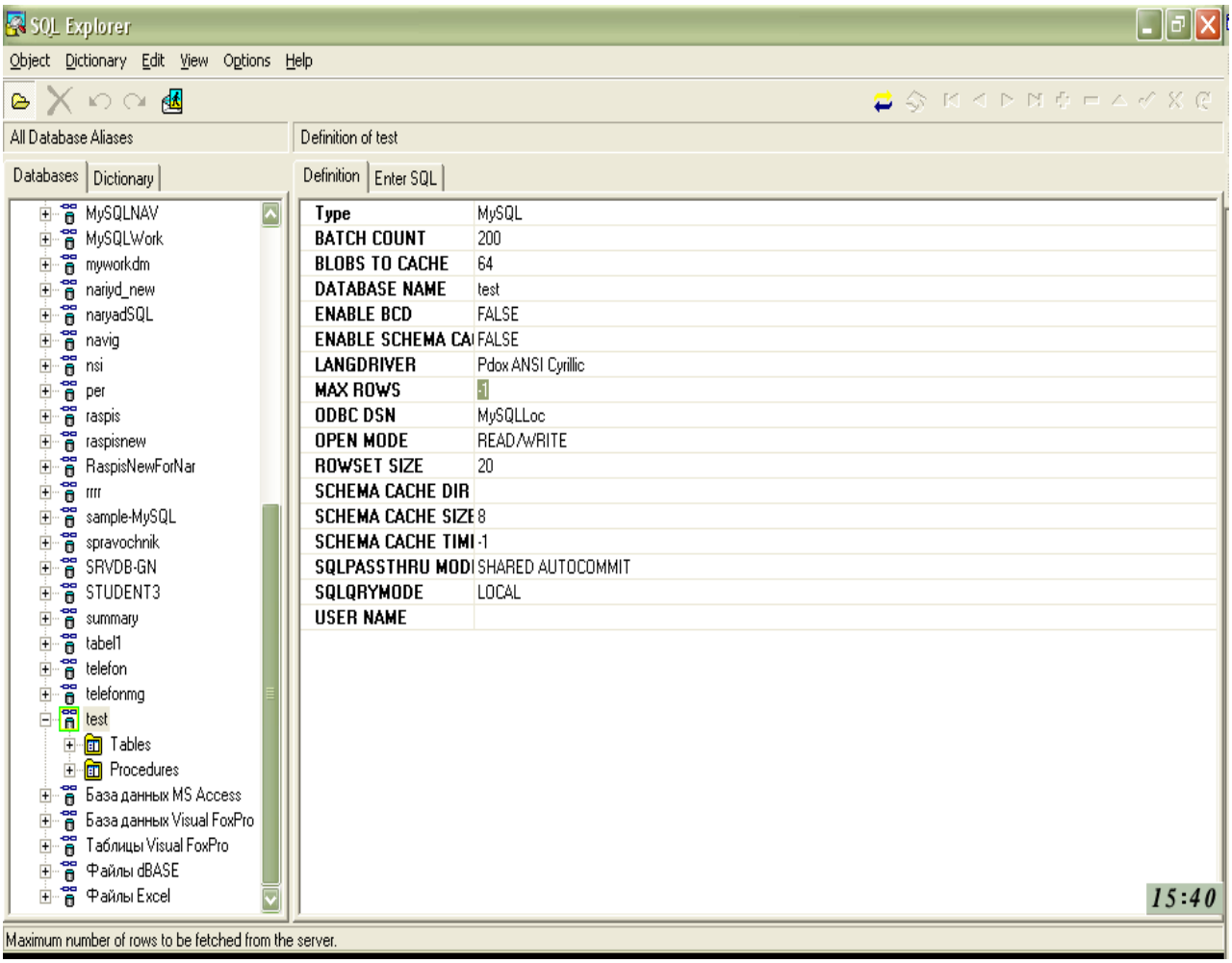

Рис. 39. Окно настройки параметров псевдонима

Драйвер источника базы данных *MySQL* устанавливается дополнительно, если нет стандартного драйвера на рабочем компьютере.

Чтобы указать источник данных *MySQL*, вызываем администратор источников данных *ODBC* (см. рис. 40). Нажимаем кнопку *Добавить* и из списка драйверов (см. рис. 41) выбираем драйвер *MySQL*. В открывшемся окне (см. рис. 42) указываем параметры: название источника данных, сервер источника данных, название базы данных, пароль подключения.

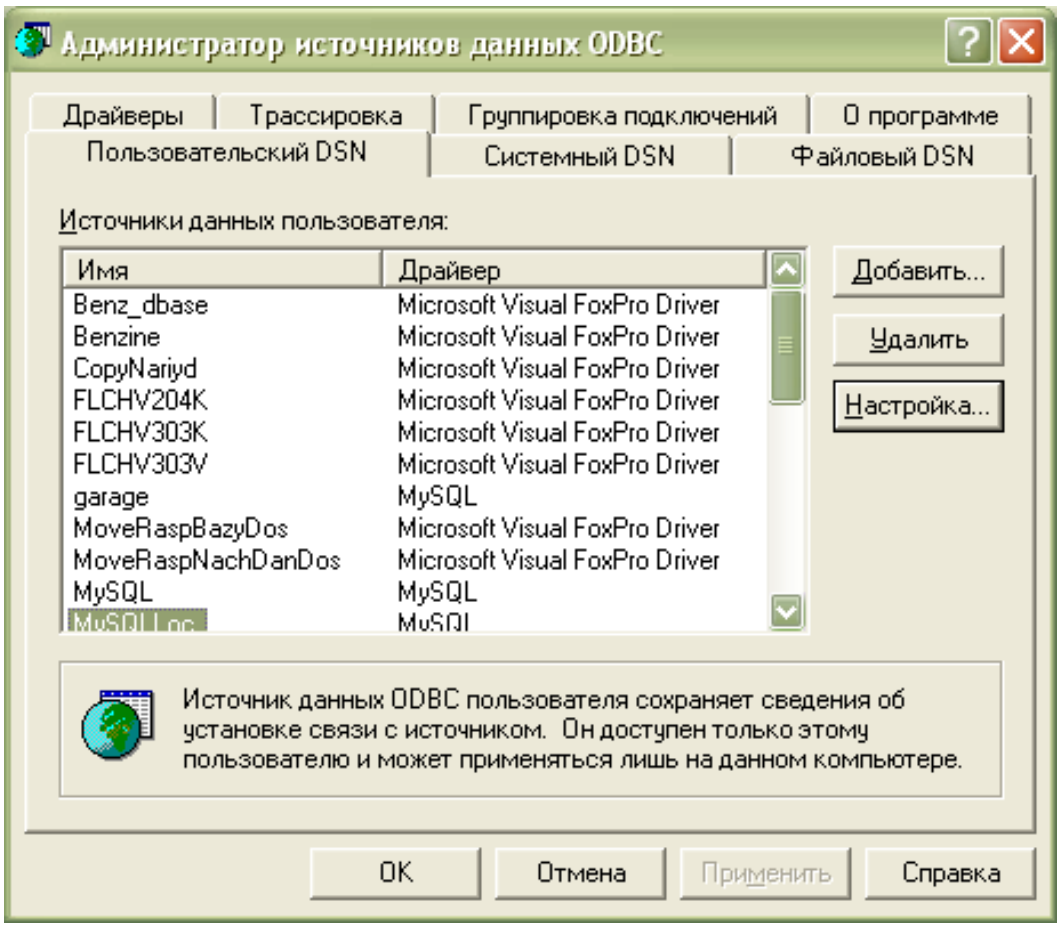

Рис. 40. Администратор источников данных *ODBC*

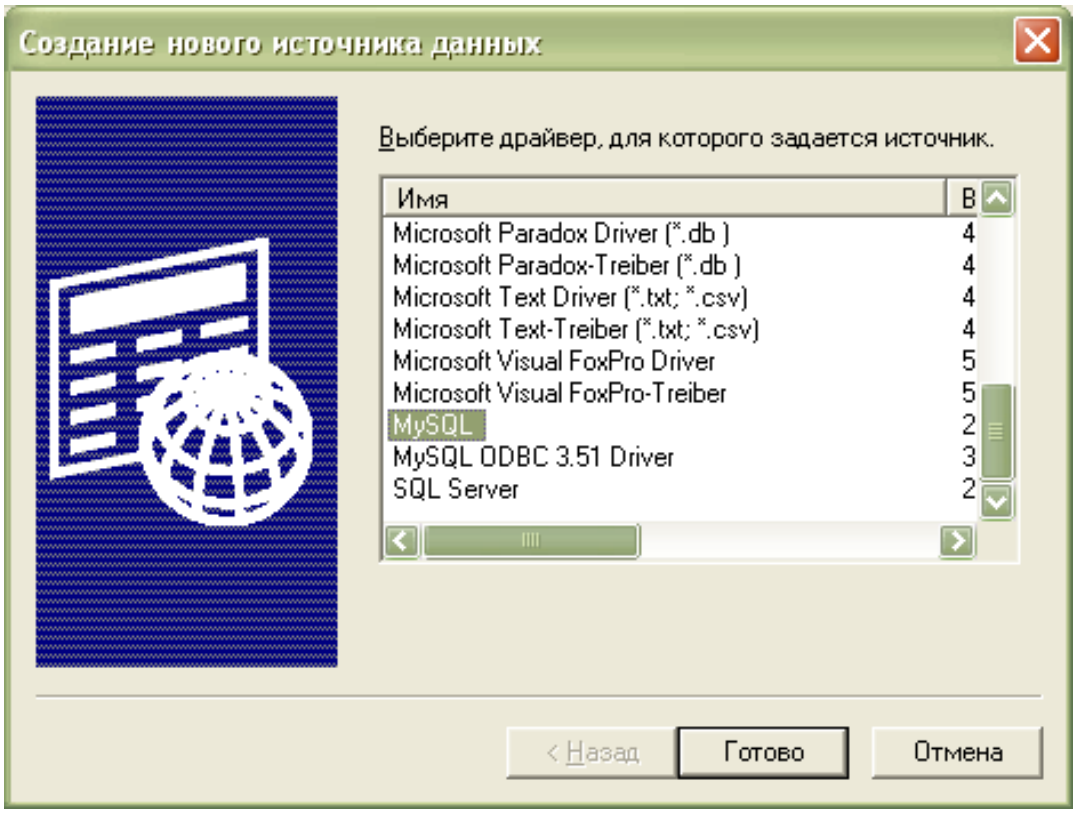

Рис. 41. Окно создания нового источника данных

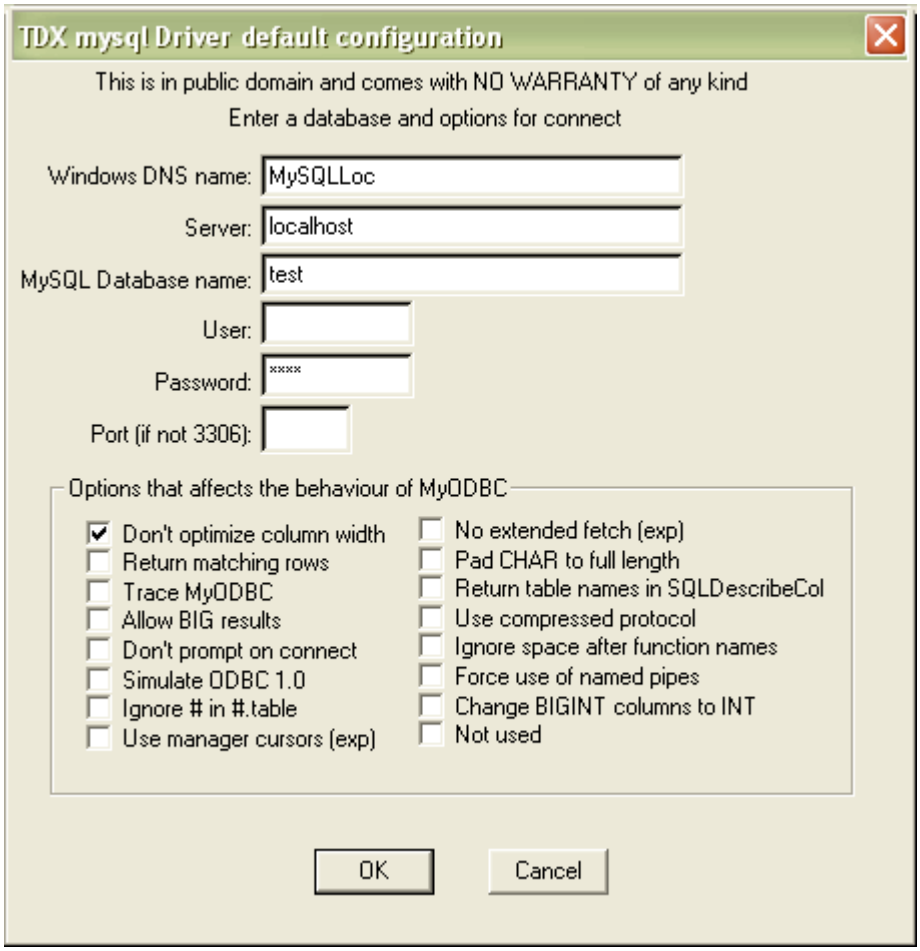

Рис. 42. Окно параметров базы данных *MySQL*

4.5. Работа со связанными таблицами

Между отдельными таблицами БД может существовать связь, которая организуется через поля таблиц. Поля связи обязательно должны быть индексированными. Связь между таблицами определяет отношение подчиненности, при котором одна таблица является главной (родительская), а другая – подчиненной (дочерняя или детальная). Обычно используется связь «один ко многим», когда одной записи в главной таблице может соответствовать несколько записей в подчиненной таблице.

Для организации связи между таблицами в подчиненной таблице используются свойства (в *Object Inspector*):

- *MasterSource* – источник данных главной таблицы;
- *IndexName* – текущий индекс;

- *IndexFieldNames* – поле или поля связи текущего индекса подчиненной таблицы;
- *MasterFields* – поле или поля индекса главной таблицы.

Для таблиц *MySQL* в качестве полей связи могут использоваться поля главного индекса, а для подчиненной таблицы – поля вторичного индекса.

Рассмотрим пример вывода данных из главной и подчиненной таблиц (рис. 43). Для организации связи в качестве поля связи главной таблицы берется автоинкрементное поле *Kod* уникального кода. По этому полю построен главный ключ, значение которого автоматически формируется при добавлении новой записи и в пределах таблицы является уникальным. В подчиненной таблице полем связи является целочисленное поле *KodPe*, по которому построен вторичный индекс.

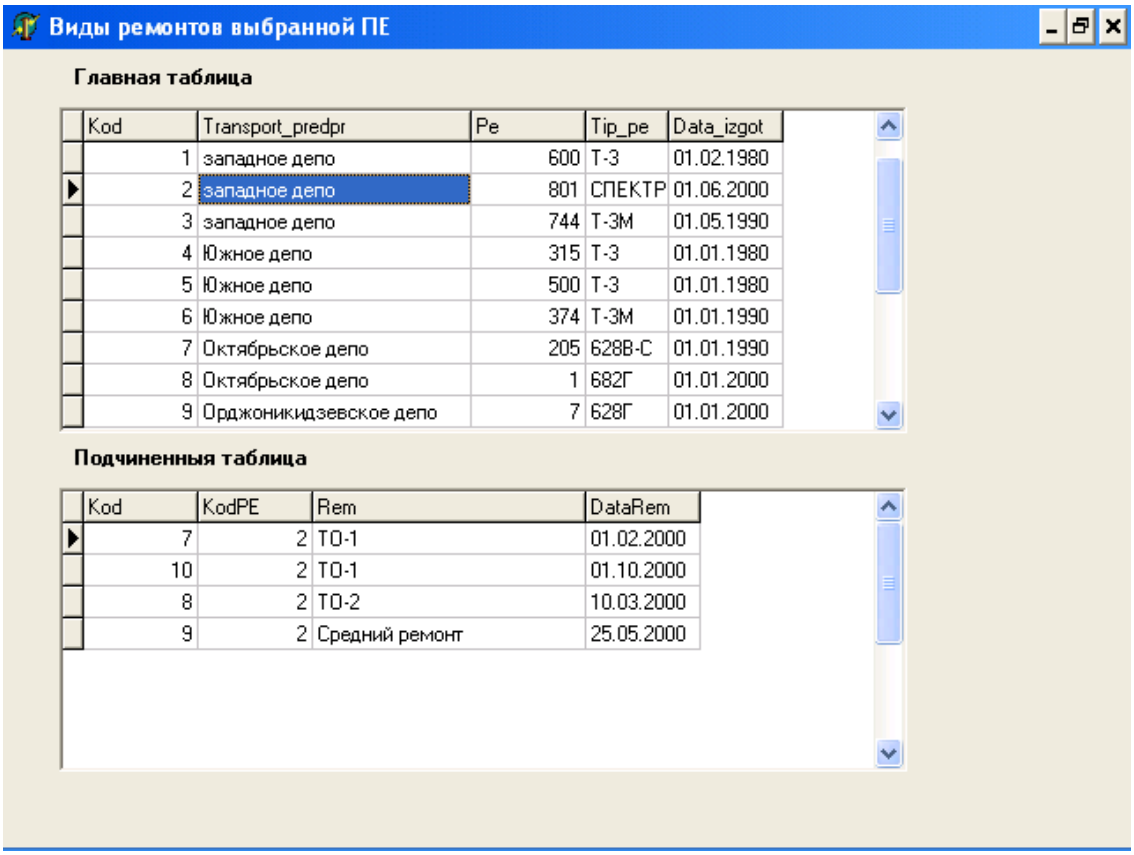

Рис. 43. Форма связи между таблицами «один ко многим»

В верхней части формы (см. рис. 43) выводится список всех подвижных единиц (ПЕ), в нижней – сведения о ремонтах выбранной ПЕ. Для наглядности в наборы данных включены все поля таблиц, которые отображаются в

компонентах *DBGrid*. При этом названия заголовков столбцов совпадают с названиями полей.

В рассмотренном примере связь между таблицами устанавливается при выполнении приложения. Обычно таблицы связываются на этапе разработки через инспектор объектов. В таком случае для установки свойства *MasterFields* можно использовать редактор полей связи *Field link Designer*. В списке *Detail Fields* выбирается поле подчиненной таблицы, а в списке *Master Fields* – поле главной таблицы (рис. 44).

После нажатия кнопки *Add* выбранные поля связываются, что отображается в списке *Joined Fields*. Заполнение свойства *MasterFields* происходит после закрытия окна при нажатии кнопки *OK*.

В *Delphi* для работы с данными таблиц применяются компоненты *Table, Query, Decision Query* и *StoredProc*.

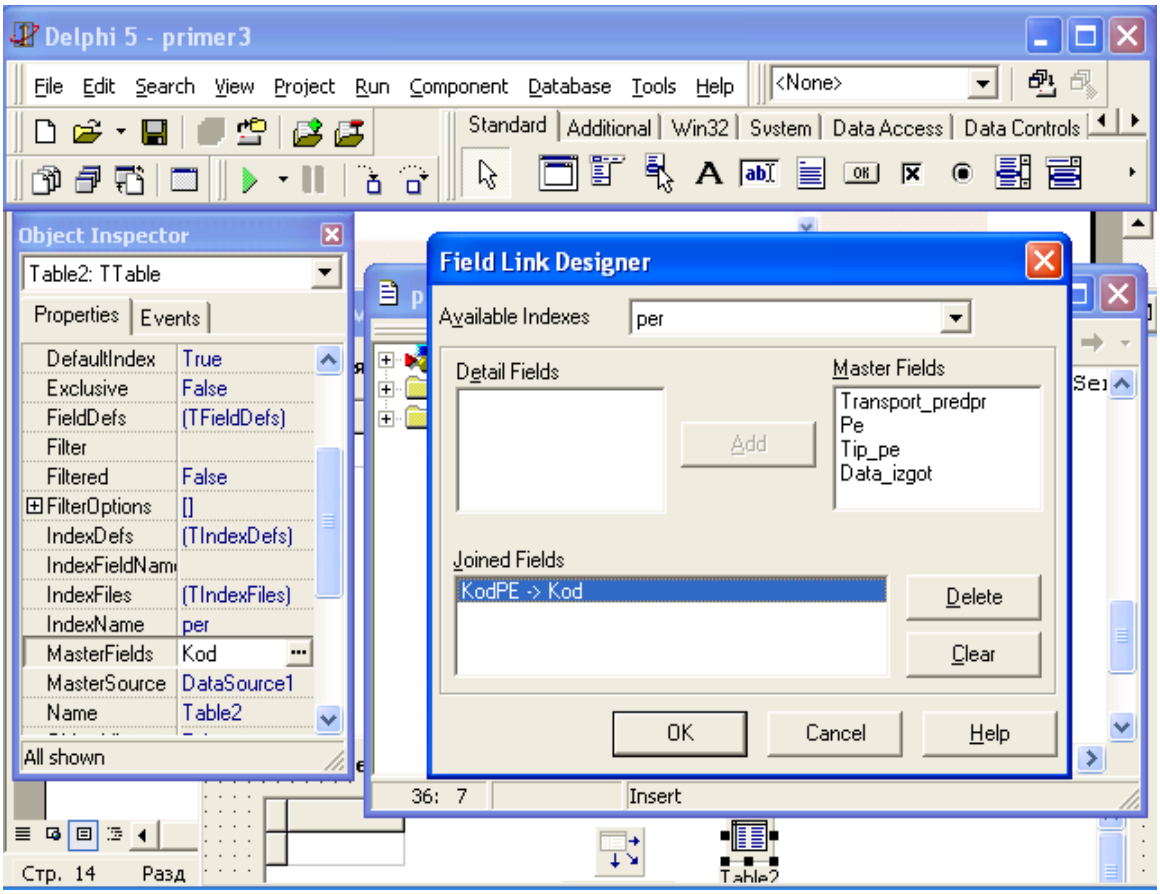

Рис. 44. Редактор полей связи

Компонент *Table* представляет собой набор данных, который в некоторый момент времени может быть связан с одной таблицей БД. Расположение БД, с таблицами которой выполняются операции, указывает свойство *DataBaseName* типа *String* (рис. 45). Значением свойства является имя каталога, в котором расположены БД, или псевдоним, ссылающийся на этот каталог.

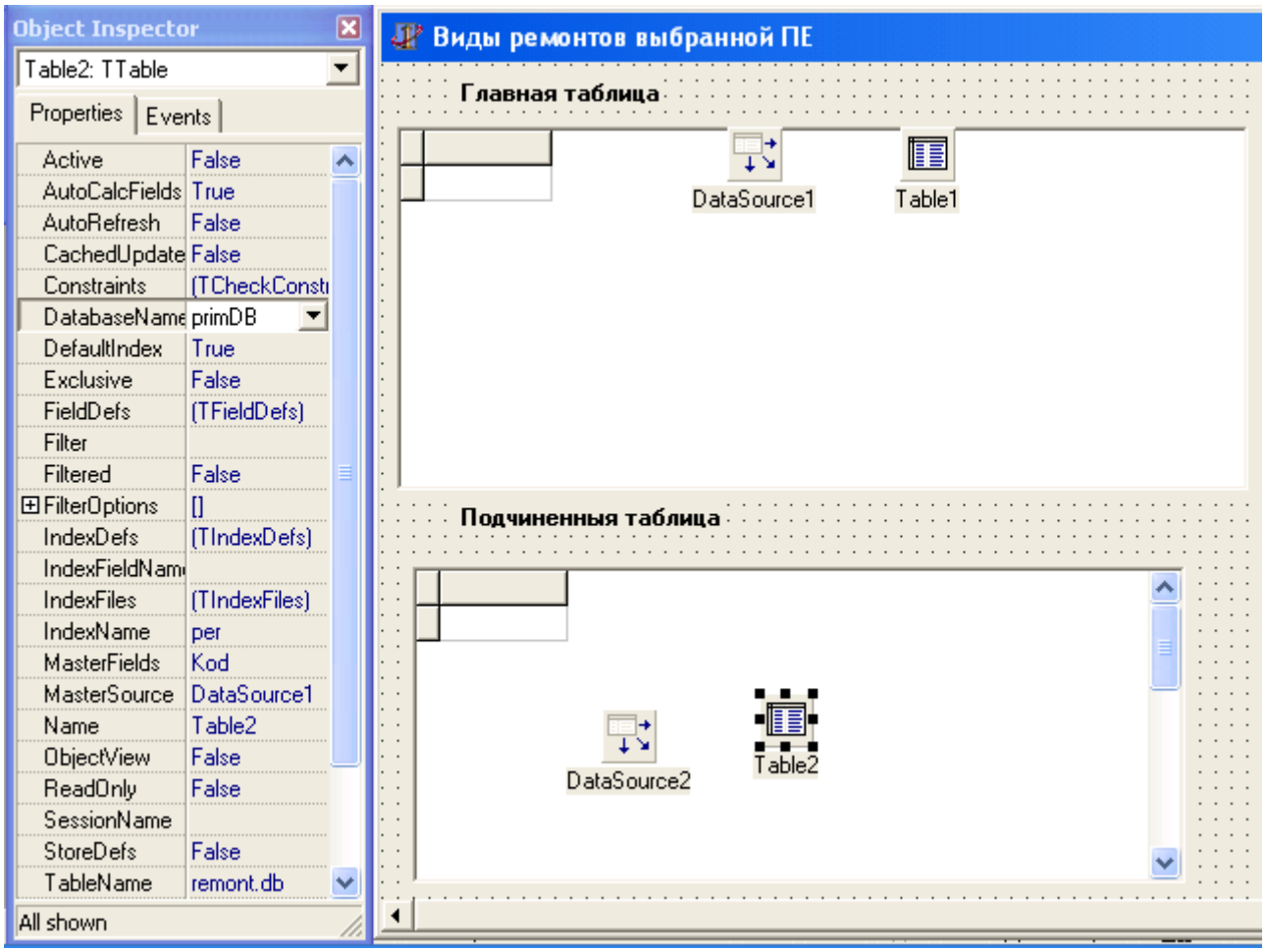

Рис. 45. Свойства компонента *Table* в инспекторе объектов

Связь между таблицей и компонентом *Table* устанавливается через его свойство *TableName* типа *TFileName*, которое определяет имя таблицы (и имя файла, содержащего данные). По умолчанию в набор данных *Table* попадают все записи связанной с ним таблицы. Для отбора данных можно использовать фильтрацию.

Для того чтобы запретить пользователю изменять записи, можно использовать свойство *ReadOnly* типа *Boolean*. По умолчанию это свойство

имеет значение *False*, которое означает, что пользователь может редактировать записи.

Установить текущий индекс можно с помощью свойства *IndexName* типа *String*. Текущий индекс выбирается из списка индексов, которые были заданы при создании таблицы. На этапе разработки приложения все возможные значения свойств *IndexName* содержат раскрывающиеся списки, доступные в инспекторе объектов.

Наборы данных могут находиться в открытом или закрытом состоянии, на что указывает свойство *Active* типа *Boolean*. Если свойству *Active* установлено значение *True*, то набор данных открыт. Если значение свойства *Active* равно *False* (по умолчанию), то набор данных закрыт и его связь с БД разорвана.

Для вывода содержимого набора данных в табличном виде удобно использовать сетку, представленную компонентом *DBGrid*. Внешний вид сетки соответствует внутренней структуре таблицы БД и набора данных, при этом строке сетки соответствует запись, а столбцу – поле.

Компонент *DataSource* – источник данных. Он используется как промежуточное звено между набором данных и визуальными компонентами, с помощью которых пользователь управляет набором данных. Для указания набора данных, с которым связан источник данных, используется свойство *DataSet* типа *TDataSet* последнего. Визуальные компоненты связаны с источником данных через свои свойства *DataSource* при проектировании в Инспекторе объектов.

Задания к лабораторной работе № 4

Цель лабораторной работы № 4 – изучить систему доступа к БД *MySQL* и научиться создавать приложение типа клиент/сервер в среде *Delphi*.

1. Ознакомьтесь с заданиями (см. задания 2, 3 в приложении). Вариант уточните у преподавателя.
2. Создайте форму приложение на языке программирования *Delphi*.
3. Организуйте доступ к БД через разработанную форму приложения.

4. Доработать форму приложения для добавления справочной и текущей информации в БД.

5. ЯЗЫК ЗАПРОСОВ БД *MYSQL*

Запуск и редактирование *SQL*-запросов осуществляется в текстовом редакторе пакета *MySQL-Front*. Редактор запросов расположен во вкладке *Query*. Выполнение запроса происходит при нажатии кнопки, размещенной вверху справа от текстового редактора *Execute SQL*…(*F*9). Результат работы запроса выводится в окне, расположенном под окном текстового редактора. Запрос дублируется в окне *SQL*, расположенном в нижней части. Здесь же показываются ошибки в тексте запроса.

Задания к лабораторной работе № 5

Цель лабораторной работы № 5 – получить практические навыки запуска и редактирования *SQL*-запросов.

1. Ознакомиться с заданиями (см. приложение). Вариант уточнить у преподавателя.
2. Выполнить *SQL*-запросы для вашего варианта (см. задания 4 ÷ 8 в приложении).
3. Оформить отчет. В отчете отразить структуру таблиц. Привести скриншот разработанной формы на *Delphi*.

6. СОЗДАНИЕ *WEB*-ОТЧЕТОВ В СРЕДЕ *DELPHI*

Специальные компоненты *Delphi* позволяют размещать информацию из БД непосредственно на *HTML*-страницы. При разработке *WEB*-приложений используются компоненты палитры компонентов, расположенных на странице *Internet* (рис. 46). Эти компоненты автоматически генерируют *HTML*-страницы.

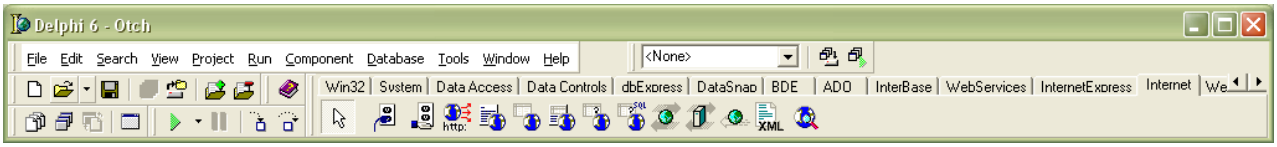

Рис. 46. Палитры компонентов на странице *Internet Delphi*6

Создайте приложение и разместите компоненты *QueryTableProducer1*, *Query1*, *DataSource1*, *Database1*, *Memo1*, *DBGrid1* (рис. 47). Настройте свойства компонента *QueryTableProducer1*: *Caption* (*название отчета*), *Columns (свойства колонок таблицы отчета)*, *Query (источник данных Query1)*. Выбрав свойство *QueryTableProducer1.Columns*, настройте колонки: *FieldName* – название поля, данные которого отображаются в этой колонке, *Title.Caption* – название колонки, *Title.Align* – центровка данных колонки, *Title.BgColor* – цвет выбранной колонки. На компонент *Button1* назначьте процедуру *Button1Click*. В этой процедуре, используя функцию *ShellExecute* из библиотеки *ShellAPI*, напишите программный код формирования *HTML*-отчета и запуска этого отчета в браузер.

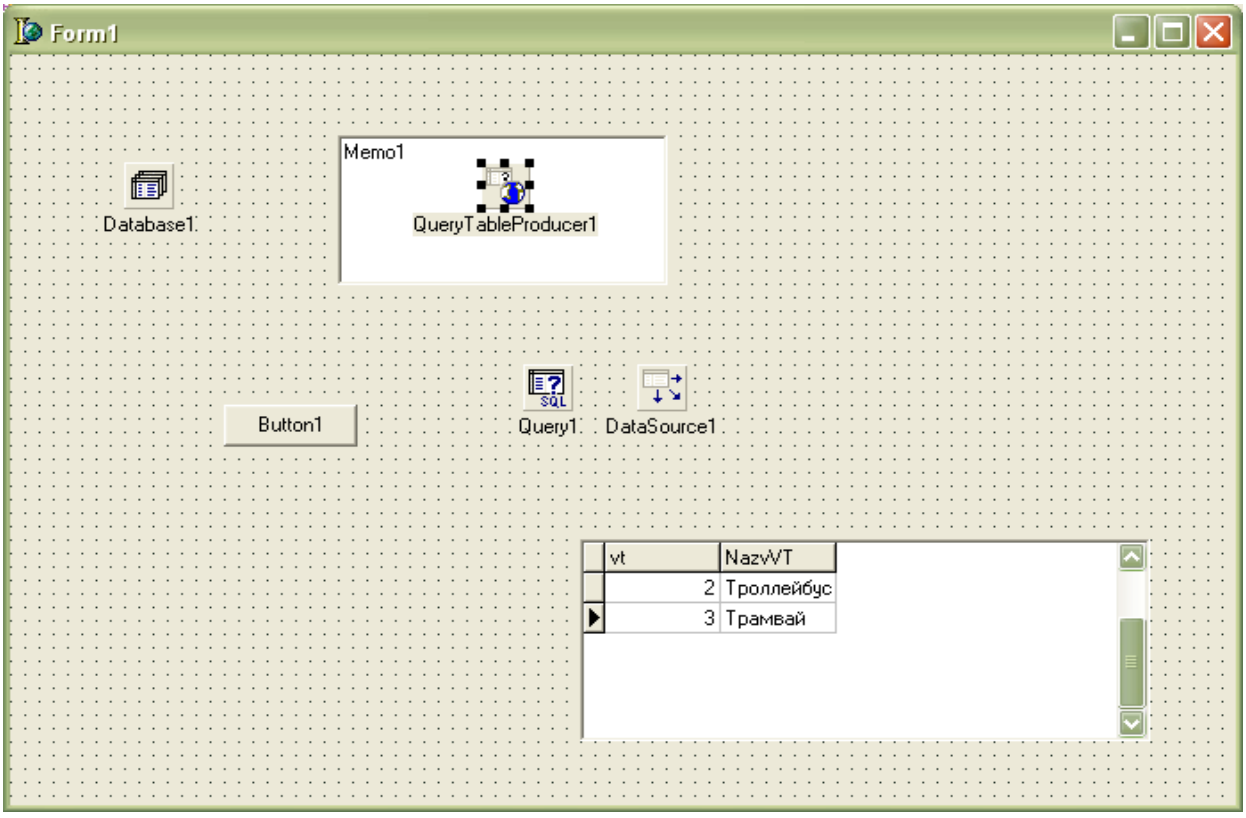

Рис. 47. Компоненты на форме приложения

Примерный листинг отчета «Вывод результатов запроса в *INTERNET EXPLORER*»:

```
unit Otch1;
interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls, HTTPApp, DBWeb,
DBBdeWeb,
  ShellAPI;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    QueryTableProducer1: TQueryTableProducer;
    Button1: TButton;
    Query1: TQuery;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Database1: TDatabase;
    procedure QueryTableProducer1GetTableCaption(Sender: TObject;
      var Caption: String; var Alignment: THTMLCaptionAlignment);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.QueryTableProducer1GetTableCaption(Sender: TObject;
  var Caption: String; var Alignment: THTMLCaptionAlignment);
begin
```

```
    Caption:=DateTimeToStr(Now())+'<br>'+'<br>'+'<br>';
  end;
  procedure TForm1.Button1Click(Sender: TObject);
  begin
      Query1.Close;
      Query1.Sql.Clear;
      Query1.SQL.ADD('select * from vt.db');
          Query1.Open;
          Query1.First;
          Memo1.Clear;
          Memo1.Text:=QueryTableProducer1.Content;
          Memo1.Lines.SaveToFile('htmpenergo.htm');
  ShellExecute(Handle,'Open',PChar('htmpenergo.htm'),'','',sw_ShowNormal);
      Query1.Close;
  end;
  end.
```

Результат выполнения листинга отчета в виде *WEB*-приложения представлен на рис. 48.

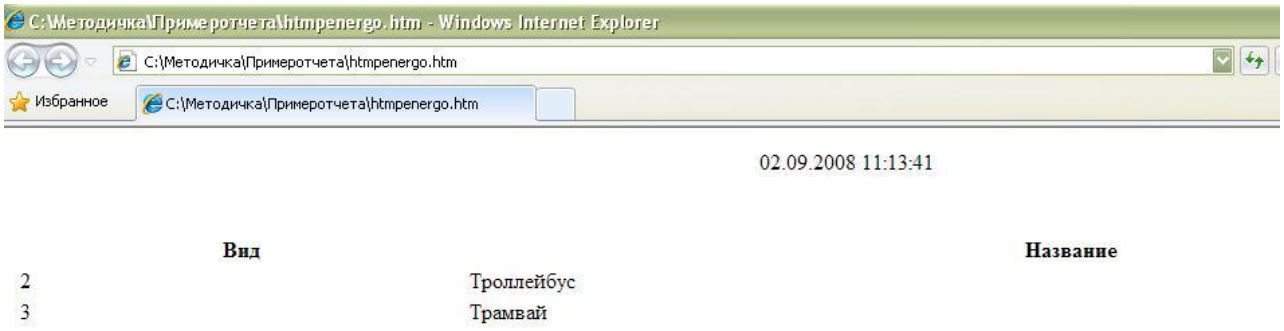

Рис. 48. Пример *WEB*-приложения

Задания к лабораторной работе № 6

Цель лабораторной работы № 6 – научиться создавать *WEB*-отчеты в среде *Delphi*.

1. Разработать *WEB*-приложение для вывода отчета в браузер для Вашего варианта задания (см. приложение п. 9).
2. Оформить отчет. В отчете отразить структуру таблиц. Привести скриншот разработанной формы на *Delphi* и отчета в браузере.

7. ОСНОВЫ *WEB*-ПРИЛОЖЕНИЙ ЯЗЫКА *PHP*

Одним из главных достоинств *PHP* является тот факт, что он внедряется прямо в *HTML*-код. *PHP* позволяет написать фрагмент следующего вида:

```
<html>
<title><? print "Hello world!"; ?></title>
</html>
```

7.1. Характеристика языка *PHP*

Для работы с *PHP* необходимо загрузить, установить и настроить *PHP* и *WEB*-сервер на компьютере. *PHP* совместим с разными *WEB*-серверами. Лучше использовать *Apache*: во-первых, это самый популярный *WEB*-сервер во-вторых, он чаще всего работает с *PHP*.

7.1.1. Переход в PHP, стандартные теги

Механизм лексического анализа должен как-то отличать код *PHP* от других элементов страницы. Идентификация кода *PHP* называется переходом в *PHP* (*escaping to PHP*). Существуют четыре варианта оформления перехода в *PHP*: стандартные теги; короткие теги; теги *script*; теги в стиле *ASP*.

Стандартные теги используются программистами *PHP* чаще остальных способов, что объясняется наглядностью и удобством этой формы записи (см. листинг 1).

Листинг 1. Вывод кода *HTML* средствами *PHP*

```
<?php print "Welcome to the world of PHP!": ?>
```

Весь текст, расположенный до закрывающего тега «?>», интерпретируется как код *PHP*.

Одной из самых замечательных особенностей *HTML* является простота использования в сочетании с другими языками, например, *HTML* и *JavaScript* (см. листинг 2).

Листинг 2. Вывод кода *HTML* средствами *PHP*

```
<html>
<head>
<title>Basic PHP/HTML integration</title>
</head>
<body>
<?
print "<h3>PHP/HTML integration is cool.</h3>";
?>
</body>
</html>
```

В листинге 3 продемонстрировано включение динамической информации в *WEB*-страницу на примере вывода текущей даты в заголовке окна.

Листинг 3. Динамический вывод даты

```
<title>PHP Recipes | <? print (date("F d, Y")); ?></title>
```

PHP позволяет изменять формат конструкций HTML. Для этого соответствующая характеристика тега присваивается переменной, вставляемой в файл. В листинге 4 эта возможность продемонстрирована на примере присваивания характеристики шрифта (*h*3) переменной *$big_font* и ее последующего использования при выводе текста.

Листинг 4. Динамические теги *HTML*

```
<html>
<head>
<title>PHP Recipes | <? print (date("F d, Y")); ?></title>
</head>
<?
$big_font = "h3";
```

```
?>
<body>
<? print "<$big_font>PHP Recipes</$big_font>"; ?>
</body>
</html>
```

7.1.2. Переменные и типы данных

Типы данных составляют основу любого языка программирования и являются средством, с помощью которого программист представляет разные типы информации. В *PHP* поддерживаются шесть основных типов данных: целые числа, вещественные числа, строки, массивы, объекты, логические величины.

Переменная представляет собой именованную область памяти, содержащую данные, с которыми можно выполнять операции во время выполнения программы.

Имена переменных всегда начинаются со знака доллара «*$*». Ниже приведены примеры допустимых имен переменных:

$color, $operating_system, $_some_variable, $model.

Имена переменных должны соответствовать тем же условиям, что и идентификаторы. Другими словами, имя переменной начинается с буквы или символа подчеркивания и состоит из букв, символов подчеркивания, цифр или других *ASCII*-символов в интервале от 127 до 255.

Переменная объявляется при первом ее использовании в программе. Тип переменной косвенно определяется по типу хранящихся в ней данных. Например:

$sentence = "This is a sentence."; // $sentence интерпретируется как строка;

$price = 42.99: // *$price* интерпретируется как вещественное число;

$weight = 185; // *$weight* интерпретируется как целое число.

Область видимости (*scope*) переменных определяется как область доступности переменной в той программе, в которой она была объявлена.

В зависимости от области видимости переменные *PHP* делятся на четыре типа: локальные переменные; параметры функций; глобальные переменные; статические переменные.

7.1.3. Параметры функций

В *PHP* любые параметры, передаваемые функции при вызове, должны быть объявлены в заголовке функции. Хотя параметрам присваиваются аргументы, переданные извне, после выхода из функции они становятся недоступными.

Параметры функции объявляются в круглых скобках после имени функции. Объявление параметров функции практически не отличается от объявления типичной переменной: функция умножает переданное значение на 10 и возвращает результат. Например:

```
function x10 ($value)
{
  $value = $value * 10;
  return $value;
}
```

7.1.4. Глобальные переменные

Глобальные переменные, в отличие от локальных, доступны в любой точке программы. Однако чтобы изменить значение глобальной переменной, необходимо специально объявить ее как глобальную в соответствующей функции. Для этого перед именем переменной ставится ключевое слово *GLOBAL*. Например:

```
$somevar = 15;
function addit()
{
  GLOBAL $somevar;
  $somevar++;
```

```
  print "Somevar is $somevar";
}
addit();
```

Будет выведено значение *$somevar*, равное 16.

7.1.5. Константы, выражения, операнды

Константой называется именованная величина, которая не изменяется в процессе выполнения программы. Константы особенно удобны при работе с заведомо постоянными величинами, например, числом π (3,141592). В *PHP* константы определяются функцией *define*(). После того как константа будет определена, будет невозможно изменить (или переопределить) ее в этой программе. Например:

```
define("'PI", "3.141592");
```

Выражение описывает некоторое действие, выполняемое в программе. Каждое выражение состоит по крайней мере из одного операнда и одного или нескольких операторов.

Операнд представляет собой некоторую величину, обрабатываемую в программе. Примеры операндов:

```
$a++;
```

где *$a* – операнд;

```
$sum = $val1 + $val2;
```

где *$sum*, *$val1* и *$val2* – операнды.

7.1.6. Операторы и операции

Операция представляет собой символическое обозначение некоторого действия, выполняемого с операндами в выражении. Операция всегда возвращает некоторое значение (результат). Оператор управляет ходом выполнения программы. Примерами операторов являются операторы выбора и цикла. В табл. 4 приведен полный список операций, упорядоченных по убыванию приоритета.

Примеры выражений:

$a = 5; – присвоить целое число 5 переменной *$a*;

$a = "5";* – присвоить строковую величину «5» переменной *$a.

Таблица 4

Операции *PHP*

Операция	Ассоциативность	Назначение
( )	-	Изменение приоритета
new	-	Создание экземпляров объектов
! ~	П	Логическое отрицание, поразрядное отрицание
++ --	П	Инкремент, декремент
@	П	Маскировка ошибок
/ * %	Л	Деление, умножение, остаток
+ - .	Л	Сложение, вычитание, конкатенация
<< >>	Л	Сдвиг влево, сдвиг вправо (поразрядный)
< <= > >=	–	Меньше, меньше или равно, больше, больше или равно
== != === <>	–	Равно, не равно, идентично, не равно
& ^ |	Л	Поразрядные операции *AND, XOR* и *OR*
&& ||	Л	Логические операции *AND* и *OR*
?:	П	Тернарный оператор
= += *= /= .= %= &= |= ^= <<= >>=	П	Операторы присваивания
AND XOR OR	Л	Логические операции *AND, XOR* и *OR*

Математические операции (табл. 5) предназначены для выполнения различных математических действий.

Таблица 5

Математические операции

Пример	Название	Результат
$a + $b	Сложение	Сумма *$a* и *$b*
$a - $b	Вычитание	Разность *$a* и *$b*
$a*$b	Умножение	Произведение *$a* и *$b*
$a/$b	Деление	Частное от деления *$a* на *$b*
$a % $b	Остаток	Остаток от деления *$a* на *$b*

Операции присваивания задают новое значение переменной (табл. 6).

Таблица 6

Операции присваивания

Пример	Название	Результат
$a = 5;	Присваивание	Переменная *$a* равна 5
$a += 5;	Сложение с присваиванием	Переменная *$a* равна сумме *$a* и 5
*$a *= 5;*	Умножение с присваиванием	Переменная *$a* равна произведению *$a* и 5
$a/=5;	Деление с присваиванием	Переменная *$a* равна частному от деления *$a* на 5
$a .= 5;	Конкатенация с присваиванием	Переменная *$a* равна конкатенации $*a* и 5

Логические операции (табл. 7) обеспечивают средства для принятия решений в зависимости от значения переменных. Логические операции позволяют управлять порядком выполнения команд в программе и часто используются в управляющих конструкциях (таких, как условная команда *if*, а также циклы *for* и *while*). Логические операции часто используются для проверки результата вызова функций:

file_exists("filename.txt") OR print "File does not exist!".

Таблица 7

Логические операции

Пример	Название	Результат
$a && $b	Конъюнкция	Истина, если истинны оба операнда
$a and $b	Конъюнкция	Истина, если истинны оба операнда
$a OR $b	Дизъюнкция	Истина, если истинен хотя бы один из операндов
!$a	Отрицание	Истина, если значение *$a* ложно
NOT !$a	Отрицание	Истина, если значение *$a* ложно
$a XOR $b	Исключающая дизъюнкция	Истина, если истинен только один из операндов

Операции равенства (табл. 8) предназначены для сравнения двух величин и проверки их эквивалентности.

Таблица 8

Операции равенства

Пример	Название	Результат
$a==$b	Проверка равенства	Истина, если *$a* и *$b* равны
$a! = $b	Проверка неравенства	Истина, если *$a* и *$b* не равны
$a===$b	Проверка идентичности	Истина, если *$a* и *$b* равны и имеют одинаковый тип

Операции сравнения (табл. 9), как и логические операции, позволяют управлять логикой программы и принимать решения при сравнении двух и более переменных. Операции сравнения предназначены для работы только с числовыми значениями.

Таблица 9

Операции сравнения

Пример	Название	Результат
1	2	3
$a<$b	Меньше	Истина, если переменная *$a* меньше *$b*

Продолжение табл. 9

1	2	3
$a>$b	Больше	Истина, если переменная *$a* больше *$b*
$a<= $b	Меньше или равно	Истина, если переменная *$a* меньше или равна *$b*
$a>= $b	Больше или равно	Истина, если переменная *$a* больше или равна *$b*
($a-12*)*?5: -1	Тернарная операция	Если переменная *$a* равна 12, то возвращается значение 5, а если не равна, то возвращается 1

К операторам выбора относится: условный оператор *(if...else)* и переключатель *(switch)*.

Синтаксис условного оператора: *if*(*condition*) *statement* 1 *else statement* 2.

Условие *condition* может быть любым выражением. Если оно истинно, то выполняется оператор *statement* 1. В противном случае выполняется оператор *statement* 2.

Переключатель *switch* является наиболее удобным средством для организации мультиветвления. Синтаксис переключателя таков:

switch(*expression*) – переключающее выражение

{

case value1: – константное выражение 1

statements; – блок операторов

break;

case value2: – константное выражение 2

statements;

break;

default:

statements;

}

Оператор *while* называется оператором цикла с предусловием. При входе в цикл вычисляется выражение условия, и, если его значение отлично от нуля, то выполняется тело цикла. Затем вычисления выражения условия и операторов тела цикла выполняются до тех пор, пока значение выражения условия не станет равным нулю, например:

```
<?
  $var = 5;
  $i = 0;
  while(++$i <= $var)
  {
    echo($i); echo('<br>');
  }
?>
```

Оператор *do...while* называется оператором цикла с постусловием. При входе в цикл в любом случае выполняется тело цикла (т. е. цикл всегда будет выполнен хотя бы один раз), затем вычисляется условие, и если оно не равно 0, то вновь выполняется тело цикла. Пример:

```
<?
  $var = 5;
  $i = 0;
  do
  {
    echo($i); echo('<br>');
  }
  while(++$i <= $var)
?>
```

Итерационный цикл имеет следующий формат:

```
  for(expression1;expression2;expression3)
  {
    statements;
```

```
}
```

Здесь *expression1* (инициализация цикла) – последовательность определений и выражений, разделяемая запятыми. Все выражения, входящие в инициализацию цикла, вычисляются только один раз при входе в цикл. Как правило, при инициализации цикла устанавливаются начальные значения счетчиков и параметров цикла, например:

```
<?
  $var = 5;
  $i = 0;
  for ($i = 0; $i <= $var; $i++)
  {
    echo($i);
    echo('<br>');
  }
?>
```

7.1.7. Определение и вызов функций

Функции могут создаваться в любой точке программ *PHP*, однако в соответствии со структурной организацией кода удобнее разместить все функции, используемые сценарием, в самом начале сценарного файла. Определение функции обычно состоит из трех частей:

- имя функции;
- круглые скобки, в которых перечисляются необязательные входные параметры, разделенные запятыми;
- тело функции, заключенное в фигурные скобки.

Обобщенный синтаксис функций *PHP* выглядит так:

```
function имя_функции ([$параметр1, $параметр2, .... $параметрN])
{
тело функции
}
```

7.1.8. Создание массивов

Массив представляет собой совокупность объектов, имеющих одинаковый размер и тип. Каждый объект в массиве называется элементом массива. При объявлении индексируемого массива после имени переменной ставится пара квадратных скобок ([ ]):

$languages[ ] = "Spanish".

Далее в массив можно добавлять новые элементы, как показано ниже. Обратите внимание: новые элементы добавляются без явного указания индекса. В этом случае новый элемент добавляется в позицию, равную длине массива плюс 1:

$languages[ ] = "English";

$languages[ ] = "Gaelic".

Функция *array*( ) получает ноль или более элементов и возвращает массив, состоящий из указанных элементов. Ее синтаксис:

array array([элемент1, элемент2...]).

Пример использования *array*( ) для создания индексируемого массива:

$languages = array ("English", "Gaelic", "Spanish").

Многомерный массив (массив массивов) – средство для хранения информации, требующее дополнительного структурирования. Создать многомерный массив несложно: добавьте дополнительную пару квадратных скобок, чтобы вывести массив в новое измерение:

$chessboard[1][4] = "King" – двухмерный массив;

$capitals["USA"]["Ohio"] = "Columbus" – двухмерный массив;

$streets["USA"]["Ohio"]["Columbus"] = "Harrison" – трехмерный массив.

В *PHP* существуют стандартные функции сортировки (табл. 10).

Таблица 10

Функции сортировки

Функция	Сортировка	Обратный порядок	Сохранение пар «ключ/значение»
sort	Значение	Нет	Нет
rsort	Значение	Да	Нет
asort	Значение	Нет	Да
arsort	Значение	Да	Да
ksort	Ключ	Нет	Да
krsort	Ключ	Да	Да
usort	Значение	?	Нет
uasort	Значение	?	Да
uksort	Ключ	?	Да

7.1.9. Простые ссылки

По ссылкам пользователь может переходить как на обычные страницы *HTML*, так и на страницы, содержащие код P*H*P:

<a href = "date.php"><"View today's date"/a>.

Если щелкнуть по ссылке, в браузере будет загружена страница с именем *date.php*.

7.2. Основные структуры *HTML* документа

7.2.1. Структура документа HTML

Документы в языке *HTML* начинаются с декларации <!*DOCTYPE*>, затем следует элемент *HTML*, внутри которого последовательно располагаются элементы *HEAD* и *BODY* (см. листинг 4):

Листинг 4. Структура документа *HTML*

<HTML>

<HEAD>

<TITLE> Текст заголовка</TITLE>

</HEAD>

<BODY>

тело документа

</BODY>

</HTML>

Минимальный документ *HTML* выглядит следующим образом (см. листинг 5):

Листинг 5. Пример *hello.html*

<TITLE>Hello</TITLE>

Результат листинга 5: *Hello world*.

7.2.2. Элемент HEAD и его производная TITLE

Пример элемента *HEAD:*

<!ELEMENT TITLE - - (#PCDATA) -(%head.misc)>*

Согласно спецификации *HTML*, каждый документ обязан иметь ровно один элемент *TITLE* в поле *HEAD*. С его помощью программе конечного пользователя сообщается название – уведомление данного документа, которое может быть выставлено в заголовке над окном соответствующей программы и т. д. Пример элемента *TITLE*:

<TITLE>Изучение динамики популяции</TITLE>.

7.2.3. Элемент BODY и его производные

Элемент *BODY* содержит собственно тело (текст) документа. В теле документа может содержаться достаточно большой набор элементов: заголовки (*H*1–*H*6), элемент *ADDRESS*, блочные элементы, элементы на уровне текста.

Элементы *H*1, *H*2, *H*3, *H*4, *H*5 и *H*6 используются в документе для разметки заголовков. У них указывают как начальный, так и конечный тэги. При этом заголовки, отмеченные элементами *H*1, главенствуют над заголовками, отмеченными элементами *H*2. Пример заголовка:

<!ELEMENT (%heading) - - (%text;)>*

<!ATTLIST (%heading) align (left|center|right) #IMPLIED >

Для элемента разметки параграфов – *P* необходимо указывать начальный тег, однако конечный тег может быть всегда опущен.

У неупорядоченных списков *UL* обязательно нужно указывать начальный и конечный тег, а также один или несколько элементов *LI*, представляющих отдельные пункты списка. В элементах *UL* и *LI* может использоваться атрибут *TYPE*, устанавливающий стиль разметки для данного списка. Неупорядоченные списки имеют вид:

<UL>

<LI> ... первый пункт списка

<LI> ... второй пункт списка

...

</UL>.

У упорядоченных (нумерованных) списков *OL* также требуется указывать начальный и конечный тег, а также один или несколько элементов *LI*, представляющих в списке отдельные пункты. Упорядоченные (нумерованные) списки имеют следующий вид:

<OL>

<LI> ... первый пункт списка

<LI> ... второй пункт списка

...

</OL>.

В списках определений *DL* требуется указывать начальный и конечный тег. В таком списке элементом *DT* отмечается термин, а элементом *DD* – соответствующее ему определение. Списки определений имеют вид:

<DL>

<DT> название термина

<DD> определение термина

...

</DL>.

Элементы *DT* можно использовать только как контейнеры для элементов текстового уровня, и в то же время внутри *DD* можно использовать блочные элементы (исключение составляют заголовки и элементы *address*). Пример списка определений:

<DL>

<DT>Первый термин<dd>Определение первого термина.

<DT>Второй термин<dd>Определение для второго термина.

</DL>.

В результате должна получиться следующая разметка:

Первый термин

Определение первого термина.

Второй термин

Определение для второго термина.

Элемент *PRE* используется, когда необходимо включить в текст документа уже отформатированный текст. Для данного элемента необходимо указывать начальный и конечный тег. Внутри такого элемента текст печатается шрифтом фиксированной ширины и сохраняется разметка оригинала (пробелы и символы конца строк). Пример использования элемента *PRE*:

<PRE>

Higher still and higher

From the earth thou springest

Like a cloud of fire;

The blue deep thou wingest,

And singing still dost soar, and soaring ever singest.

</PRE>

будет воспроизведено как:

Higher still and higher

From the earth thou springest

Like a cloud of fire;

The blue deep thou wingest,

And singing still dost soar, and soaring ever singest.

DIV – деление документа на отдельные блоки. Для данного элемента необходимо указывать начальный и конечный тег.

CENTER – выравнивание текста. Для этого элемента необходимо указывать начальный и конечный тег. Используется для центрирования стоящего между ними текста.

FORM – заполняемые формы. Необходимо указывать начальный и конечный тег. Используется для создания заполняемой формы, которая будет обрабатываться на *HTTP* сервере.

HR – горизонтальные линейки. Элемент не является контейнером, так что использовать конечный тег нельзя.

TABLE – таблица, может быть вложенной.

В общем случае таблицы имеют следующий вид:

<TABLE BORDER=3 CELLSPACING=2 CELLPADDING=2 WIDTH="80%">

<CAPTION> ... заголовок таблицы ... </CAPTION>

<TR><TD> первая клетка таблицы </TD><TD> вторая клетка </TD>

</TR> ...

</TABLE>

Для элемента *TABLE* всегда необходимо указывать начальный и конечный тег. При этом разрешается использовать следующие атрибуты:

align – принимает одно из следующих значений – *LEFT*, *CENTER* или *RIGHT* (используемый при этом регистр значения не имеет). Указывает для текущей таблицы, каким образом при разметке осуществляется ее горизонтальное выравнивание;

width – в отсутствии данного атрибута ширина таблицы определяется автоматически;

border – позволяет задавать для таблицы ширину внешней рамки в пикселах (например, *BORDER* = 4). Данному атрибуту может быть также присвоено значение нуль, чтобы полностью отказаться от внешней рамки;

cellspacing – в языке *HTML* каждая ячейка имеет собственную границу, отделенную промежутком от границ соседних ячеек;

cellpadding – устанавливает для каждой ячейки в таблице расстояние в пикселах между рамкой ячейки и содержащимся в ней материалом.

Элемент *CAPTION* может иметь только один атрибут – *ALIGN*, который может принимать два значения: *ALIGN = TOP* или *ALIGN = BOTTOM*.

Для *TR* элемента, начинающего новый ряд таблицы, необходимо указывать начальный тег, однако всегда можно опустить конечный тег. Элемент *TR* выступает в роли контейнера для ячеек таблицы и может иметь два атрибута:

align – устанавливает стиль горизонтального выравнивания для содержимого ячейки, который будет использоваться по умолчанию. Атрибут принимает одно из следующих значений (независимо от используемого регистра): *LEFT, CENTER* или *RIGHT* – и выполняет ту же самую роль, что и атрибут *ALIGN* при разметке параграфов;

valign – используется при выборе правила, согласно которому – если нет других указаний – будет осуществляться вертикальное выравнивание во всех ячейках данной строки. Атрибут может принимать одно из следующих значений (независимо от используемого регистра): *TOP, BOTTOM* или *MIDDLE*. При этом содержимое ячейки будет выравниваться по ее верхнему или нижнему краю либо посередине.

7.2.4. Элементы текстового уровня

Для всех элементов, которые задают шрифт, используемый при разметке документа, требуется указывать начальный и конечный тег, например:

<B>полужирный текст</B>.

<B>полужирный и <I>текст с курсивом</I></B>.

Для форматирования текста в *HTML* используют следующие атрибуты:

TT – телетайпный текст или текст фиксированной ширины;

I – стиль шрифта с курсивом;

B – стиль с полужирным шрифтом;

U – стиль с подчеркиванием текста;

STRIKE – стиль с перечеркиванием текста;

BIG – печать текста шрифтом большего кегля;

SMALL – печать текста шрифтом меньшего кегля;

SUB – печать текста со сдвигом вниз (нижний индекс);

SUP – печать текста со сдвигом вверх (верхний индекс).

В спецификации *HTML* документа применяют следующие элементы разметки фраз:

EM – основной элемент, используемый в HTML для выделения текстов, обычно реализуется в виде курсива;

STRONG – усиленное выделение, обычно реализуемое посредством полужирного шрифта;

DFN – выделяет описываемый термин;

CODE – используемый для выделения программного кода;

SAMP – используемый при предоставлении в документе распечаток программ, программных сценариев и т. д.;

KBD – используемый для выделения текста, который пользователь должен набрать на клавиатуре;

VAR – используемый для обозначения переменных либо аргументов, передаваемых с командами;

CITE – используемый для обозначения цитат или ссылок на другие источники.

7.3. *FORM* (форма) – заполняемая форма

7.3.1. Атрибуты формы

Форма используется для таких действий пользователя, как регистрация, упорядочение пользователя или формирование запроса. Основной синтаксис формы и ее возможные атрибуты представлены в табл. 11:

<FORM ACTION="URL">

Таблица 11

Возможные атрибуты формы

Имя атрибута	Возможные значения	Смысл атрибута	Примечания
ACTION	*URL*	Адрес сервера, который использует форма	Сервер *HTTP* или *URL*
METHOD	*GET, POST*	Метод передачи данных, полученных от пользователя, на сервер	По умолчанию – *GET*
ENCTYPE	Строка	Механизм, используемый для кодирования содержимого формы	По умолчанию приложение */x-www-form-url-кодирование*

Элементы, которые могут появиться только в пределах элемента *FORM*:

INPUT – текстовое однострокoвое поле, поля пароля, переключатели, радиокнопки, кнопки установки и перезагрузки, скрытые поля, кнопки выгрузки файла, кнопки изображений и т. д.;

SELECT – меню единичного или множественного выбора;

TEXTAREA – многострочное текстовое поле.

7.3.2. Способы обработки данных

PHP создает следующие массивы при обработке данных, пришедших из формы:

$_GET – *GET*-параметры, переданные к скрипту через переменную окружения *QUERY_STRING*. Например, *$_GET('login')*;

$_POST – данные формы, полученные с помощью метода *POST*;

$_COOKIE – все *cookies*, которые прислал браузер;

$_REQUEST – объединение всех перечисленных выше массивов. Чтобы жестко не «привязываться» к типу принимаемых данных (*GET* или *POST*) именно эту переменную и рекомендуется использовать в скриптах;

$_SERVER – переменные окружения, переданные сервером.

При отладке сценария можно использовать переменную *$GLOBALS*, например, как в листинге 6:

Листинг 6. Файл отладки.

```
<!-- Выводит все глобальные переменные -->
<pre>
<? Print_r ($GLOBALS); ?>
</pre>
```

7.3.3. Передача данных командной строкой

Пусть на сервере в корневом каталоге есть сценарий *PHP* под названием *test_forma.php* (см. листинг 7, рис. 49). Сценарий распознает два параметра: *login* и *password*.

Листинг 7. Страница с формой *test_forma.php*.

```
<html><body>
<?php
echo '<form action=test_pw.php>
Логин: <input type="text" name="login" value="root"><br>
Пароль: <input type="password" name="password" value="zz"><br>
<input type="submit" value="Нажмите кнопку">
</form>';
?>
</body></html>
```

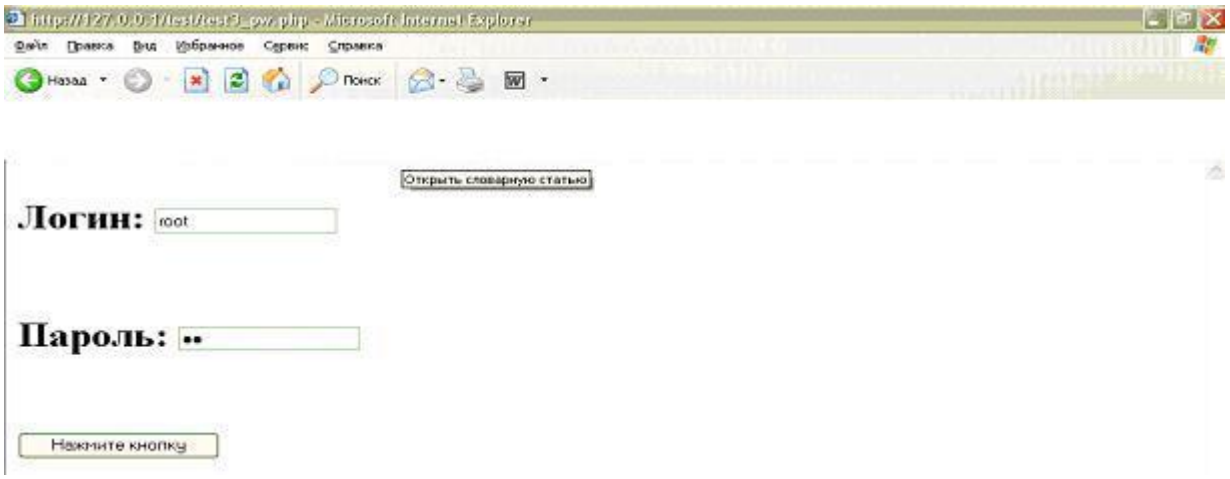

Рис. 49. Результат выполнения листинга 7

Если в адресной строке браузера задать http://localhost/hello.php?login=root&password=zz, то получается передача параметров в командной строке.

Проанализировать переменную окружения *$QUERY_STRING*, которая доступна в *PHP*, можно под именем *$_SERVER[QUERY_STRING]* (см. листинг 8, рис. 50).

Листинг 8. Файл проверки параметров командной строки *test_pw.php*.

```
<html><body>
<?php
 Echo '<b><h1>Данные из командной строки:,<br><br>';
 echo $_SERVER[QUERY_STRING],'</h1></b><br>';
?>
</body></html>
```

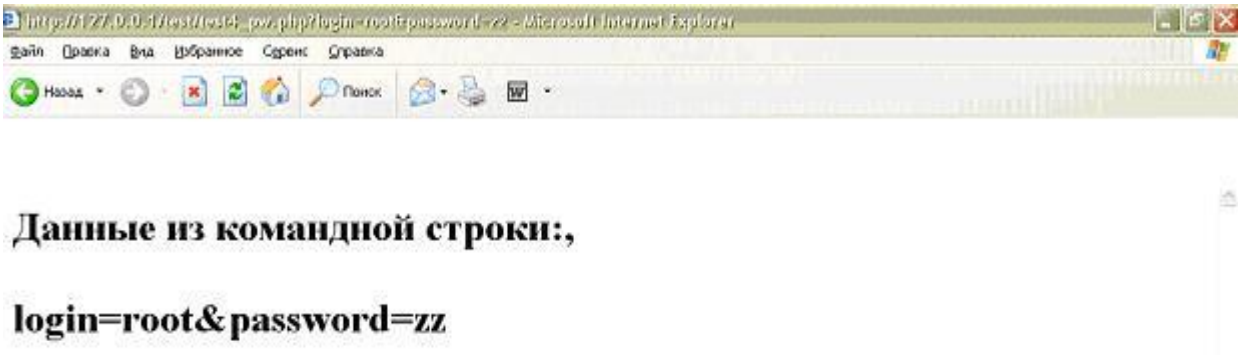

Рис. 50. Результат выполнения листинга 8

7.3.4. Трансляция полей формы

Все данные из полей формы *PHP* помещает в глобальный массив *$_REQUEST*. Значение поля *login* будет храниться в *$_REQUEST*[‘*login’*], а значение поля *password* – в *$_REQUEST*[‘*password*] (см. листинг 9, файл *test_pw.php*).

Листинг 9. Использование данных форм.

```
<html><body>
<?php
 if ($_REQUEST['login']=='root' && $_REQUEST['password']=='zz') {
        echo "Доступ открыт для пользователя – ";
        echo $_REQUEST['login'];
 } else {
        echo "Доступ закрыт";
 }
?>
</body></html>
```

7.3.5. Трансляция переменных окружения

Переменные окружения можно преобразовать в локальные переменные (листинг 10).

Листинг 10. Вывод *IP*-адреса и браузера пользователя.

```
<html><body>
<?php
 echo "Ваш IP-адрес:", $_SERVER['REMOTE_ADDR'],"<br>";
 echo "Ваш браузер:", $_SERVER['HTTP_USER_AGENT'];
?>
</body></html>
```

7.3.6. Трансляция cookies

Все переменные, переданные скрипту, создают массив (листинг 11).

Листинг 11. Демонстрация работы с *$_COOKIES*.

```
<?php
// С начала счетчик равен нулю
$count=0;
// Если в cookies что-то есть, берем счетчик оттуда.
If (isset ($_COOKIE["count"])){
 $count=($_COOKIE["count"]);
 //echo "count==",$count;
 $count++;
 setcookie("count", $count,0x7FFFFFFF, "/");
}
// выводит счетчик
Echo "КУКА= ",$count;
?>
```

7.3.7. Обработка списков

В *PHP* предусмотрена возможность задавать имена полям формы в виде массива с индексами (листинг 12, рис. 51):

Листинг 12. Обработка списков.

```
<html>
   <head>
      <title>Пример</title>
   </head>
   <body>
      <?php
      echo '<FORM ACTION="test6_sel.php" METHOD=POST>
                  Список городов:
                  <SELECT NAME=gorod[]>
                  <OPTION>Екатеринбург
                  <OPTION>Москва
```

```
            <OPTION>Санкт-Петербург
            <OPTION>Киев
            </SELECT>
            <P>
            <INPUT TYPE=SUBMIT VALUE="Вывод">
        </FORM>';
 ?>
  </body>
</html>
```

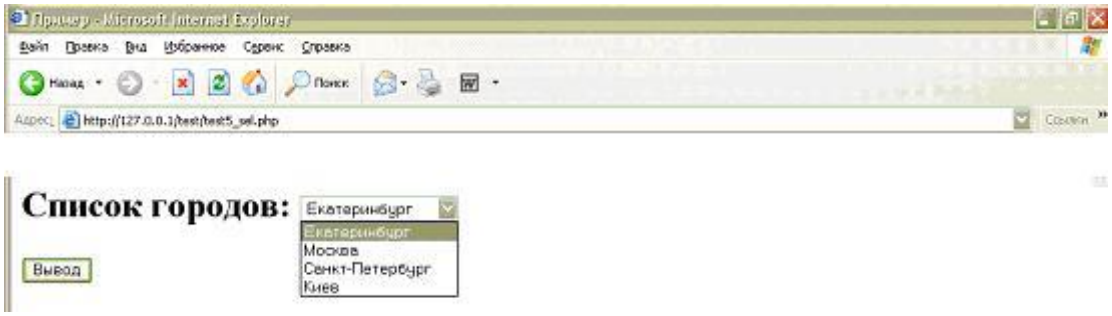

Рис. 51. Результат выполнения листинга 12

В результате выполнения листинга 12 получим массив в *$_REQUEST* массив массивов (двумерный), доступ к элементам которого можно получить так (листинг 13):

Листинг 13. *test6_sel.php.*

```
<html>
   <head>
      <title>Пример</title>
   </head>
   <body>
      <?php
         $gor=$_REQUEST['gorod'][0];
         Echo $gor;  ?>
    </body>
</html
```

Пример ассоциативного массива представлен в файле *test7_sel.php* (листинг 14, рис. 52):

Листинг 14. Файл *test7_sel.php*.

```
<html>
  <head>
      <title>Пример</title>
  </head>
  <body>
    <?php
      echo '<FORM ACTION="test8_sel.php" METHOD=POST>
      Имя <input type=text name= Data[name]><br>
      Адрес <input type=text name= Data[address]><br>
      Город
      <input type=radio name= Data[city] value=Moscow>Москва<br>
      <input type=radio name= Data[city] value=Peter>Петербург<br>
      <input type=radio name= Data[city] value=Kiev>Киев<br>
      <INPUT TYPE=SUBMIT VALUE="Вывод">
      </FORM>';
  ?>
  </body>
</html>
```

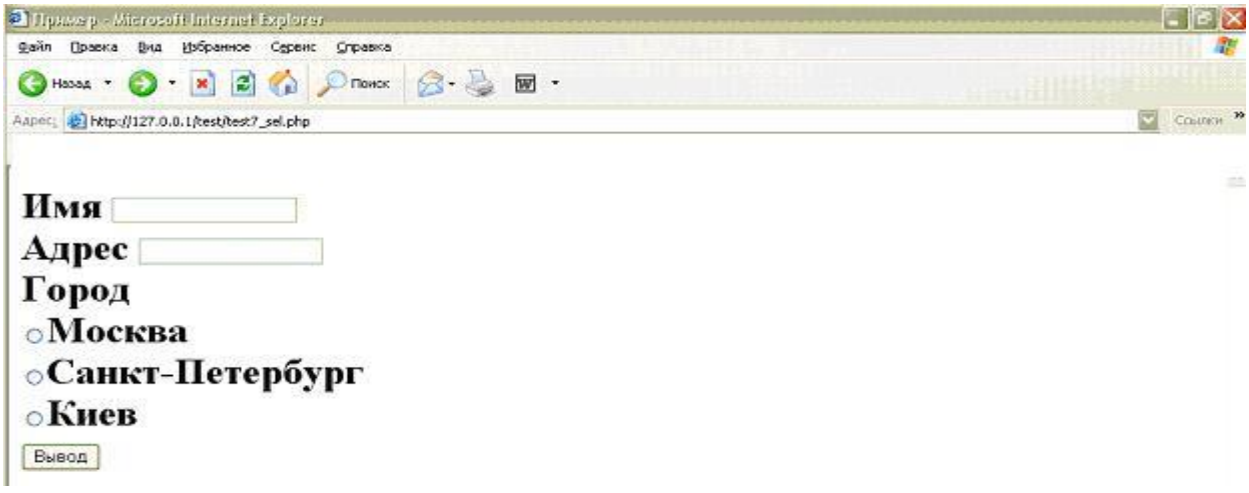

Рис. 52. Результат выполнения листинга 14

В сценарии к отдельным элементам формы можно обратиться при помощи указания ключа массива: например, *$REQUEST[Data][city]* (листинг 15):

Листинг 15. Файл *test8_sel.php*.

```
<html>
  <head>
    <title>Пример</title>
  </head>
  <body>
    <?php
          $gor=$_REQUEST[Data][city];
          Echo $gor;
          echo '<br>',"Имя - ";
          echo $_REQUEST[Data][name];
 ?>
  </body>
</html>
```

7.3.8. Передача параметров методами GET и POST

При использовании метода *GET* все параметры передаются единой строкой в переменной *QUERY_STRING*. Данные поступят *URL*-кодированными (см. подробнее в пп. 7.4, листинг 17).

При использовании метода *POST* параметры передаются сценарию через стандартный поток ввода (см. листинги 12, 14).

7.3.9. Особенности флажков checkbox

Можно воспользоваться одноименным скрытым полем (*hidden*) со значением, равным, например, нулю, поместив его перед нужным флажком (см. листинг 16, рис. 53).

Листинг 16. Гарантированный прием значений от флажков в файле *test12_hid.php*.

```
<html>
 <head>
   <title>Пример</title>
</head>
<body>
 <?php
        foreach (@$_REQUEST[known] as $k=>$v) {
              If ($v) {echo "Вы знаете язык $k!"."<br>";}
              else {echo "Вы не знаете язык $k!"."<br>";}
        }
 echo '<form action=test12_hid.php method=POST>
 какие языки программирования вы знаете?<br>
 <input type=hidden name="known[PHP]" value="0">
 <input type=checkbox name="known[PHP]" value="1">PHP<br>
 <input type=hidden name="known[Perl]" value="0">
 <input type=checkbox name="known[Perl]" value="1">Perl<br>
 <input type=submit name="doGo" value="doGo">
 </form>';
 ?>
</body>
</html>
```

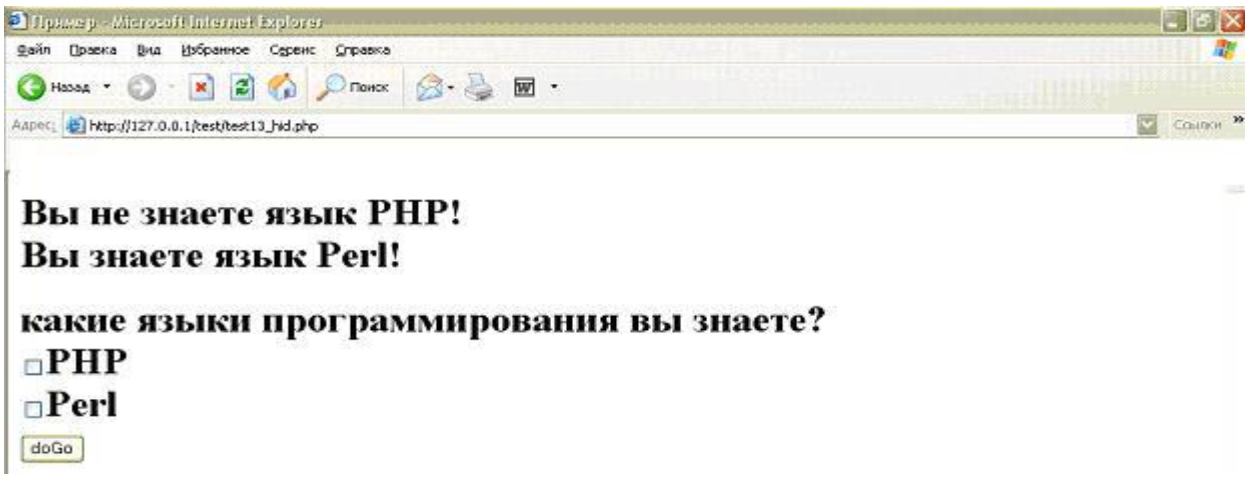

Рис. 53. Результат выполнения листинга 16

Если пользователь не выберет никакой флажок, то браузер отправит сценарию пару *known*[язык] = 0 и в массиве *$_REQUEST*[*known*] создастся соответствующий элемент. Если пользователь выберет флажок, то последует пара *know*[язык] = 1.

7.4. Пример формы

От описания базовых компонентов форм мы переходим к практическому примеру – построению формы для обработки данных, введенных пользователем. Допустим, вы хотите создать форму выбора периода времени (рис. 54).

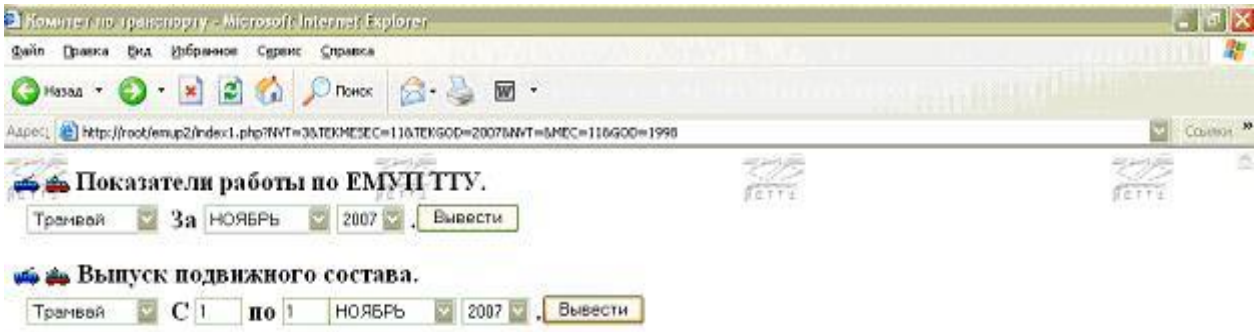

Рис. 54. Результат выполнения листинга 17

Программа *index1.php* приведена в листинге 17.

Листинг 17. Программа *index1.php*

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>Комитет по транспорту</title>
</head>
<body bgcolor="#ffffff" BACKGROUND="images/fon.jpg">
<?php
$DEN=date( 'd')+0; – объявление значений переменных
$MEC=date( 'm');
$GOD=date( 'Y');
```

Echo '<FORM METHOD="get" ACTION="Pokrab/pokrabttu_bas.php" target="_top">';

Echo '<IMG SRC="/image/troll.gif" BORDER=0 ALIGN=CENTER>';

Echo " ";

Echo "<B>Показатели работы по ЕМУП ТТУ.</B>";

*Echo "
  ";*

include ("../vvt2.php"); – список видов транспорта

Echo "<B> 3a ";

// месяцы года

Echo '<SELECT NAME="TEKMESEC">';

Echo '<OPTION ';if ($MEC==1) {Echo 'SELECTED';} Echo ' VALUE="01"> ЯНВАРЬ</OPTION>';

Echo '<OPTION ';if ($MEC==2) {Echo 'SELECTED';} Echo ' VALUE="02"> ФЕВРАЛЬ</OPTION>';

Echo '<OPTION ';if ($MEC==3) {Echo 'SELECTED';} Echo ' VALUE="03"> МАРТ</OPTION>';

Echo '<OPTION ';if ($MEC==4) {Echo 'SELECTED';} Echo ' VALUE="04"> АПРЕЛЬ</OPTION>';

Echo '<OPTION ';if ($MEC==5) {Echo 'SELECTED';} Echo ' VALUE="05"> МАЙ</OPTION>';

Echo '<OPTION ';if ($MEC==6) {Echo 'SELECTED';} Echo ' VALUE="06"> ИЮНЬ</OPTION>';

Echo '<OPTION ';if ($MEC==7) {Echo 'SELECTED';} Echo ' VALUE="07"> ИЮЛЬ</OPTION>';

Echo '<OPTION ';if ($MEC==8) {Echo 'SELECTED';} Echo ' VALUE="08"> АВГУСТ</OPTION>';

Echo '<OPTION ';if ($MEC==9) {Echo 'SELECTED';} Echo ' VALUE="09"> СЕНТЯБРЬ</OPTION>';

Echo '<OPTION ';if ($MEC==10) {Echo 'SELECTED';} Echo ' VALUE="10"> ОКТЯБРЬ</OPTION>';

Echo '<OPTION ';if ($MEC==11) {Echo 'SELECTED';} Echo ' VALUE="11"> НОЯБРЬ</OPTION>';

Echo '<OPTION ';if ($MEC==12) {Echo 'SELECTED';} Echo ' VALUE="12"> ДЕКАБРЬ</OPTION>';

Echo '</SELECT> ';

$ii = 1999; // годы

Echo '<SELECT NAME="TEKGOD">';

$i = $GOD;

while ($i >= $ii) {

Echo '<OPTION ';if ($GOD==$i) {Echo 'SELECTED ';}

Echo 'VALUE="'.$i.'">'.$i.'</OPTION>';

$i--;

}

Echo '</SELECT> ';

Echo '</B><INPUT TYPE="SUBMIT" VALUE="Вывести">';

Echo "</FORM>";

?>.

Задания к лабораторной работе № 7

Цель лабораторной работы № 7 – получить практические навыки написания *PHP*-отчетов и *WEB*-программирования. Познакомиться с основами языка программирования *PHP*.

1. Откройте пакет *XAMPP Control Panel*. Ярлык на рабочем столе или в папке *C:/xampp/xampp-control.exe*. Запустите локальный *Web*-сервер *Apache* для *Windows* и *MySQL*.
2. Создайте первый сайт с расширением **.php*, используя пакет *NotePad++*. Файл сохраните в папке, например, *gruppa*1 директории *C:/xampp/htdocs/*. Запустите его из адресной строки браузера, например, по доменному адресу: *http://localhost/gruppa1/primer1.php* или по *IP*-адресу:

http://127.0.0.1/gruppa1/primer1.php. При отладке скрипта в браузере для обновления сайта используйте клавишу *F*5 или кнопку *Обновить*.

3. Создайте второй сайт из предложенного задания. Все данные из полей формы *PHP* поместите в глобальный массив *$_REQUEST*.
4. Отчет по лабораторной работе должен содержать:

- текст *PHP*-кода программы;
- результат выполнения программы – рисунки двух получившихся сайтов;
- выводы о проделанной работе.

Варианты заданий к лабораторной работе № 7

Вариант 1

Напишите *PHP*-программу создания формы «Ввод простого текста и его вывод».

На форме расположите:

- поле ввода (используйте тег <*input*>, параметр *type=text*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

В поле ввода внесите фамилию, имя и отчество студента. На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме введите:

- в первой строке фамилию студента полужирным шрифтом;
- во второй строке его имя курсивом;
- в третьей строке фамилию и инициалы студента.

Примечание. Функция длины строки *strlen*(*string* $*st*) *int* возвращает количество символов в строке $*st*; функция поиска подстроки *strpos*(*string* $*where*, *string* $*what*, *int* $*from*=0) *int* возвращает целочисленное значение позиции строки $*where*, с которой начинается подстрока $*what*; функция вывода подстроки *substr*(*string* $*str*, *int* $*start* [,*int* $*length*]) *string* возвращает участок строки $*str*, начиная с позиции строки $*start* и длиной $*length*.

Вариант 2

Напишите *PHP*-программу создания формы «Ввод пароля и вывод проверки – правильно ли веден пароль».

На форме расположите:

- поле ввода (используйте тег <*input*>, параметр *type=password*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название кнопки *OK*.

На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится сообщение о проверке правильности введенного пароля. На экране должно выйти сообщение «Пароль введен верно» или «Пароль введен неверно». Это сообщение должно выводиться полужирным шрифтом.

Вариант 3

Напишите *PHP*-программу создания формы «Выбор параметра независимого переключателя и вывод соответствующего значения».

На форме расположите:

- независимый переключатель (или флажок) (используйте тег <*input*>, параметр *type=checkbox* поле ввода);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

Значения переключателя: «текущее время», «текущая дата», «текущий день», «текущий месяц», «текущий год». На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выведите выбранное значение. Этот параметр определяется в соответствии с выбором переключателя.

Примечание. Функция текущей даты – *date*(*"d.m.y"*), функция времени – *date*(*"h.i.s"*).

Вариант 4

Напишите *PHP*-программу создания формы «Выбор параметра независимого переключателя и вывод соответствующего значения цвета».

На форме расположите:

- независимый переключатель (или радиокнопки) (используйте тег <*input*>, параметр *type=radio*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

Значения переключателя – «красный цвет», «желтый цвет», «синий цвет», «зеленый цвет». На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится текст названия цвета в соответствующей цветовой гамме, посередине формы.

Примечание. Используйте тег *FONT,* который имеет следующий синтаксис: <*FONT SIZE=n*>*текст*</*FONT*> или <*FONT COLOR=цветовая спецификация*>*текст*</*FONT*>. Для переноса строки используется тег <*br*>.

Вариант 5

Напишите *PHP*-программу создания формы «Вставка картинки и вывод даты или времени».

На форме расположите два рисунка для отправки формы (*image*) (используйте тег <*input*>, параметр *type=image*).

На первый рисунок наложите программу вывода текущего времени, на второй рисунок – программу вывода текущей даты. Перед значением текущей даты (например, 01.05.2012 г.) напишите «Сегодня 1 мая 2012 года». Перед текущим временем напишите «Текущее время» в верхнем регистре.

Примечание. Функция *strtoupper*(*string* $*st*) *string* переводит строку в верхний регистр.

Вариант 6

Напишите *PHP*-программу создания формы «Создание формы выбора даты и вывод этой даты».

На форме расположите:

- поле ввода значения числа дня (используйте тег <*input*>, параметр *type=text*);
- раскрывающийся список *select* значения названия месяца (используйте атрибуты <*name*>, <*option*> *и value*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится выбранная дата (например, 1 мая 2012 года).

Вариант 7

Напишите *PHP*-программу создания формы «Созданные формы выбора натурального числа и вывод квадрата или куба этого числа».

На форме расположите:

- раскрывающийся список *select* значения названия натурального числа с 1 до 10 (используйте атрибуты <*name*>, <*option*> и *value*);
- независимый переключатель (или радиокнопки) (используйте тег <*input*>, параметр *type=radio*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

На радиокнопку наложите два параметра: квадрат, куб. На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится квадрат или куб выбранного натурального числа соответственно выбранному параметру радиокнопки.

Примечание. Для возведения в квадрат используйте функцию *pow(float $base, float $exp). float* возвращает $*base* в степень $*expl*. Для возведения в куб используйте цикл *for…end*.

Вариант 8

Напишите *PHP*-программу создания формы «Ввод интервала простых чисел и вывод случайного числа из этого интервала».

На форме расположите:

- два поля ввода (используйте тег <*input*>, параметр *type=text*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

В первом поле ввода набивается начальное значение интервала, во втором поле ввода – конечное значение интервала. На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится случайное число.

Примечание. Функция возвращения случайных чисел – *mt_rand(int $min=0, int $max=RAND_MAX) int.*

Вариант 9

Напишите *PHP*-программу создания формы «Создание массива фамилий и демонстрация работы со списками».

На форме расположите кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится список фамилий. Список пронумеруйте.

Примечание. Для вывода используйте цикл *for*. Количество элементов в массиве определяется функцией *count()*. Элементы массива пропишите в *PHP*-программе.

Вариант 10

Напишите *PHP*-программу создания формы «Ввод интервала простых чисел и вывод списка заданных значений».

На форме расположите:

- два поля ввода (используйте тег <*input*>, параметр *type=text*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

В первом поле ввода набивается начальное значение интервала, во втором поле ввода – конечное значение интервала. На кнопку *Вывести* наложите еще

одну *PHP*-программу, с помощью которой в новой форме выводится столбиком заданный интервал чисел, начиная со второго значения и заканчивая предпоследним значением.

Примечание. Для вывода чисел напишите функцию вывода.

Вариант 11

Напишите *PHP*-программу создания формы «Ввод значений любых чисел и вывод значения».

На форме расположите:

- два поля ввода (используйте тег <*input*>, параметр *type=text*);
- кнопку отправки формы *submit* (используйте тег <*input*>, параметр *type=submit* и название *Вывести*).

В первом и во втором поле ввода вносятся дробное значение. На кнопку *Вывести* наложите еще одну *PHP*-программу, с помощью которой в новой форме выводится результат перемножения этих чисел, а также выведите: ближайшее целое число, наименьшее целое число, максимальное целое число и абсолютное значение.

Примечание. Функция *abs(mixed $numeric) mixed* возвращает модуль числа. Функция *round(double $val) double* округляет $*val* до ближайшего целого числа. Функция *ceil(float $number) int* возвращает наименьшее целое число, которое было бы не меньше, чем *number*. Функция *float(float $number) int* возвращает максимальное целое число, не превосходящее *number*.

8. РАЗРАБОТКА *WEB*-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ *PHP* ДЛЯ БД *MYSQL*

8.1. Стандартные функции *PHP* для работы с *MySQL*

Самая значимая возможность в *PHP* – наличие интеграции с БД.

С помощью стандартных функций *PHP* можно организовать взаимодействие сценариев *PHP* с сервером *MySQL* (см. листинг 18).

Последовательность действий при взаимодействии с сервером *MySQL* выглядит так:

1. установите соединение с сервером *MySQL*. Если попытка завершается неудачей, то необходимо вывести соответствующее сообщение и завершить процесс;
2. выберите базу данных сервера *MySQL*. Если попытка выбора завершается неудачей, то надо вывести соответствующее сообщение и завершить процесс. Допускается одновременно открыть несколько баз данных для обработки запросов;
3. обработайте запросы к выбранной базе (или базам);
4. закройте соединение с сервером БД.

Листинг 18. Пример сценария *PHP* с базой данных.

```
<?php
$sock=mysql_connect('localhost', 'nobody', 'sasnobody');
mysql_select_db('disr',$sock);
$SQL='
SELECT VT AS TRP, VIDTRAN AS ID FROM disr.NACHDAN where vt>1 GROUP BY VT ORDER BY VT DESC
';
$R=mysql_query($SQL,$sock);
$T=mysql_fetch_array($R);
Echo '<SELECT NAME='NVT'>';
while(is_array($T))
{
Echo ' <OPTION VALUE='.$T[TRP]. ' >'.$T[ID];
$T=mysql_fetch_array($R);
}
Echo '</SELECT>';
$sock=mysql_close;
?>.
```

8.1.1. Mysql_connect()

Функция *mysql_connect*() устанавливает связь с сервером *MySQL*. После успешного подключения к *MySQL* можно переходить к выбору баз данных, обслуживаемых этим сервером. Синтаксис функции:

mysql_connect ([string хост [:порт] [:/путь//к/сокету] [, string имя пользователя] [, string пароль])

В параметре функции *хост* передается имя сервера БД *MySQL*. Оно же используется для перенаправления запросов на *web*-сервер, на котором работает *MySQL*, поскольку к серверу *MySQL* можно подключаться в удаленном режиме. Необязательные параметры: *номер порта*, *путь к сокету* (для локального компьютера). Параметры *имя_пользователя* и *пароль* должны соответствовать имени пользователя и паролю, заданным в таблицах привилегий *MySQL*.

Пример открытия соединения с *MySQL*:

@mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server!').

В данном примере *localhost* – имя компьютера, *nobody* – имя пользователя, а *sasnobody* – пароль. Знак @ перед вызовом функции *mysql_connect*() подавляет все сообщения об ошибках, выдаваемые при неудачной попытке подключения, – они заменяются сообщением, указанным при вызове *die*().

8.1.2. Mysql_select_db()

После успешного соединения с *MySQL* необходимо выбрать базу данных, находящуюся на сервере. Для этого используется функция *mysql_select_db()*. Синтаксис функции:

mysql_select_db (string имя_базы_данных [, int идентификатор_соединения]).

Параметр *имя_базы_данных* определяет выбираемую базу данных и идентификатор, который возвращается функцией *mysql_select_db()*.

Пример выбора базы данных функцией:

<?php

$sock=mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server!');

mysql_select_db('godar',$sock) or die('Could not select company database!');

?>

8.1.3. Mysql_close()

После завершения работы с сервером *MySQL* соединение необходимо закрыть. Функция *mysql_close()* закрывает соединение, определяемое необязательным параметром. Если параметр не задан, то функция закрывает последнее открытое соединение. Синтаксис функции:

mysql_close ([int идентификатор_соединения]).

Пример использования *mysql_close*():

<?php

$sock=mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server!');

mysql_select_db('godar',$sock) or die('Could not select company database!');

print 'You're connected to a MySQL database!';

$sock=mysql_close();

?>

8.1.4. Mysql_query()

Функция *mysql_query()* обеспечивает интерфейс для обращения запросов к БД. Синтаксис функции:

mysql_query (string запрос [, int идентификатор_соединения]).

Параметр *запрос* содержит текст запроса на языке *SQL*. Если обработка запроса завершилась неудачей, то функция возвращает *FALSE*. Пример использования функции:

$SQL='

select TN,FIO,CLASS from nariyd_new.FIO

Where TP=301 *And TIP*=1*';*

$Result=mysql_query($SQL,$sock);

8.1.5. Mmysql_num_rows()

Функция *mysql_num_rows()* определяет количество записей, возвращаемых командой *SELECT*. Синтаксис функции*:*

int mysql_num_rows(int результат).

Пример использования функции *mysql_num_rows()*:

<?php

// Подключиться к серверу и выбрать базу данных

@mysql_connect('localhost', 'web', '4tf9zzzf') or die('Could not connect to MySQL server!');

@mysql_select_db('company') or die('Could not select company database!');

// Выбрать все товары, названия которых начинаются с *'p'*

$query = 'SELECT prod_name FROM products WHERE prod_name LIKE

'p%'';

// Выполнить запрос

$result = mysql_query($query);

print 'Total rows selected: '.mysql_num_rows($result);

$sock=mysql_close();

?>

Поскольку таблица содержит лишь один товар, название которого начинается с буквы *p* (*pears*), постольку возвращается только одна запись. Результат:

Total rows selected: 1

8.1.6. Mysql_result()

Функция *mysql_result()* используется в сочетании с *mysql_query()* (при выполнении запроса с командой *SELECT*) для получения набора данных.

Синтаксис функции:

mysql_result (int идентификатор_результата, int запись [, mixed поле]).

В параметре *идентификатор_результата* передается значение, возвращенное функцией *mysql_query()*. Параметр *запись* ссылается на определенную запись набора данных, определяемого параметром *идентификатор_результата*.

В необязательном параметре *поле* могут передаваться следующие данные:

- смещение поля в таблице;
- имя поля;
- имя поля в формате *имя_поля_имя_таблицы*.

В листинге 19 используется БД *nariyd_new* и таблица *FIO* (фамилии).

Листинг 19. Выборка и форматирование данных в базе данных *MySQL*.

```
<?php
$sock=mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server! ');
mysql_select_db('godar',$sock) or die('Could not select company database!');
$SQL='
  select TN,FIO,CLASS from nariyd_new.FIO
  Where TP=301 And TIP=1';
$Result=mysql_query($SQL,$sock);
$kolzap=mysql_numrows($Result);
Echo 'Список фамилий';
for ($j=0;$j<=$kolzap-1;$j++)
{
  Echo $j+1, '.';
  Echo mysql_result($RF,$j, 'FIO')). ' ;';
  Echo '<br>';
}
$sock=mysql_close();
?>
```

В результате выполнения этого примера будет получен результат:

Список фамилий

Иванов;

Петров;

Синицына;

8.1.7. Mysql_fetch_row()

Обычно гораздо удобнее сразу присвоить значения всех полей записи элементам индексируемого массива (начиная с индекса 0), нежели многократно вызывать *mysql_result()* для получения отдельных полей. Для этого используется функция чтения массива данных *mysql_fetch_row(),* имеющая следующий синтаксис:

array mysql_fetch_row (int результат).

Использование функции *list*() в сочетании с *mysql_fetch_row()* позволяет сэкономить несколько команд, необходимых при использовании *mysql_result().* В листинге 20 приведен код листинга 19, переписанный с использованием *list*() и *mysql_fetch_row().*

Листинг 20. Выборка данных функцией *mysql_fetch_row().*

```
<?php
$sock=mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server! ');
mysql_select_db('godar',$sock) or die('Could not select company database! ');
$SQL='
  select TN,FIO,CLASS from nariyd_new.FIO
  Where TP=301 And TIP=1';
$Result=mysql_query($SQL,$sock);
$i=0;
while ($row = mysql_fetch_row ($Result));
  Echo $i++;
  Print $row['FIO'], ';', '<br>';
```

```
  $row++;
endwhile;
$sock=mysql_close();
?>
```

Листинг 20 выдает тот же результат, что и листинг 19, но использует при этом меньшее количество команд.

8.1.8. Mysql_fetch_array()

Функция *mysql_fetch_array()* аналогична *mysql_fetch_row(),* по умолчанию значения полей записи сохраняются в ассоциативном массиве. Можно выбрать тип индексации (ассоциативная, числовая или комбинированная). Синтаксис функции:

array mysql_fetch_array (int идентификатор результата [, тип_индексации]).

В параметре *идентификатор_результата* передается значение, возвращенное функцией *mysql_query().* Необязательный параметр *тип_индексации* принимает одно из следующих значений:

MYSQL_ASSOC – функция *mysql_fetch_array*() возвращает ассоциативный массив. Если параметр не указан, то данное значение используется по умолчанию;

MYSQL_NUM – функция *mysql_fetch_array*() возвращает массив с числовой индексацией;

MYSQL_BOTH – к полям возвращаемой записи можно обращаться как по числовым, так и по ассоциативным индексам.

Листинг 21 содержит вариант кода листингов 19 и 20. Используется функция *mysql_fetch_array*(), возвращающая ассоциативный массив полей.

Листинг 21. Выборка данных функцией *mysql_fetch_array().*

```
<?php
$sock=mysql_connect('localhost', 'nobody', 'sasnobody') or die('Could not connect to MySQL server! ');
```

```
mysql_select_db('godar',$sock) or die('Could not select company database!');
$SQL='
  select TN,FIO,CLASS from nariyd_new.FIO
  Where TP=301 And TIP=1';
$Result=mysql_query($SQL,$sock);
$i=0;
while($row = mysql_fetch_array($Result))
{
  Echo $i++;
  Print $row['FIO'], ';', '<br>';
}
mysql_free_result($result);
?>
```

Листинг 21 выдает тот же результат, что и листинги 19 и 20.

8.1.9. Mysql_free_result()

Функция *mysql_free_result()* освобождает память для запроса. Синтаксис функции:

int mysql_free_result(int result).

Функция *mysql_free_result()* используется только в том случае, если требуется большой объем памяти во время работы скрипта. Вся используемая память для идентификатора запроса будет освобождена автоматически.

8.1.10. Mysql_field_name, mysql_field_table, mysql_field_type, mysql_field_len

Функция *mysql_field_name* получает имя указанного поля в запросе. Синтаксис функции:

string mysql_field_name(int result, int field_index).

Аргументами функции являются идентификатор запроса и индекс поля.

Функция *mysql_field_table* получает имя таблицы, в которой находится указанное поле.

Синтаксис функции:

string mysql_field_table(int result, int field_offset).

Функция *mysql_field_type* получает тип указанного поля в запросе. Синтаксис функции:

string mysql_field_type(int result, int field_offset).

Функция подобна функции *mysql_field_name()*. Второй аргумент функции возвращается тип поля, например *int*, *real*, *string*, *blob*. Пример – в листинге 22.

Листинг 22

```
<?php
mysql_connect('localhost:3306');
mysql_select_db('wisconsin');
$result = mysql_query('SELECT * FROM onek');
$fields = mysql_num_fields($result);
$rows = mysql_num_rows($result);
$i = 0;
$table = mysql_field_table($result, $i);
echo 'Your''.$table. ''table has'.$fields. 'fields and'.$rows 'records <BR>';
echo 'The table has the following fields <BR>';
while ($i < $fields)
{
   $type  = mysql_field_type  ($result, $i);
   $name  = mysql_field_name  ($result, $i);
   $len   = mysql_field_len   ($result, $i);
   $flags = mysql_field_flags ($result, $i);
   echo $type. ''.$name. ''.$len. ''.$flags. '<BR>';
   $i++;
}
$sock=mysql_close();
?>
```

Функция *mysql_field_len* возвращает длину указанного поля. Синтаксис функции:

int mysql_field_len(int result, int field_offset).

8.1.11. Mysql_list_fields

Функция *mysql_list_fields* показывает список полей *MySQL* в запросе. Синтаксис функции:

int mysql_list_fields(string database_name, string table_name, int [link_identifier]).

Функция *mysql_list_fields()* извлекает информацию о заданной *tablename* таблице. Аргументы – имя базы данных и имя таблицы. После выполнения функции возвращается указатель запроса, который может использоваться функциями *mysql_field_flags(), mysql_field_len(), mysql_field_name(),* и *mysql_field_type()*. Идентификатор запроса является положительным целым. Функция возвращает -1, если происходит ошибка. Строка описания ошибки будет помещена в переменную $*phperrmsg*, и если функция не была вызвана как *@mysql()*, то на экран будет выведена эта же строка описания ошибки.

8.1.12. Mysql_num_fields

Функция *mysql_num_fields* получает количество полей в запросе. Синтаксис функции:

int mysql_num_fields(int result).

8.2. Пример вывода таблицы в форму и выбора параметра

Пример формы обработки таблицы из базы данных приведен в листинге 23, результат листинга показан на рис. 55.

Листинг 23. Файл *test*15*_sql.php*.

```
<html>
  <head>
    <title>Пример</title>
```

```
    </head>
  <body>
  <?php
  $sock=mysql_connect('localhost', 'nobody', '');//соединение с сервером
  mysql_select_db('test', $sock);// соединение с БД test
  $SQL='SELECT kod, name from test.syrnal'; // SQL запрос
  $R=mysql_query($SQL,$sock);// обработка SQL запроса
  echo  '<h1><FORM  ACTION='test16_sql.php'  METHOD=POST>';  //  форма
передачи параметров
  echo 'Список журналов:';
  echo '<SELECT NAME=surnal[]>'; // объявление переменной
  echo '<OPTION value=0>Все';
  if ($R) {
    $Tcol=mysql_NumRows($R); // количество записей в запросе
    if ($Tcol)>0{
          for ($i=0;$i<=$Tcol-1;$i++) { // обработка запроса по записям
              $p1=mysql_fetch_array($R);
              echo '<OPTION value='.$p1[kod]. '>'.$p1[name]; // вывод
                 каждой записи
          }
    }
  }
  echo '</SELECT>';
  echo '<INPUT TYPE=SUBMIT VALUE="Вывод">'; // вывод результата
  echo '</FORM></h1>';
  $sock=mysql_close; // закрытие БД
  ?>
  </body>
  </html>
```

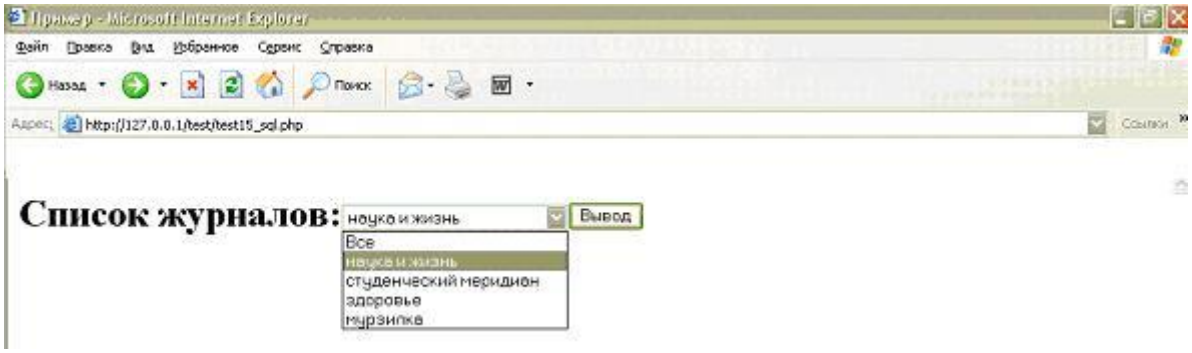

Рис. 55. Результат выполнения листинга 23

После нажатия кнопки *Вывод*, выводится скрипт *test*16*_sql.php* (см. листинг 24 и рис. 56).

Листинг 24. Текст программы *test*16*_sql.php*.

```
<html>
   <head>
      <title>Список журналов</title>
   </head>
<body>
<?php
$sock=mysql_connect('localhost', 'nobody', ''); //соединение с сервером
mysql_select_db('test',$sock);// соединение с БД 'test'
$sur=$_REQUEST['surnal'][0]; // обработка переменной
$SQL='SELECT kod, name from test.syrnal'; // SQL запрос из БД 'test'
if ($sur<>0) {$SQL.= 'where kod='.$sur;}
$R=mysql_query($SQL,$sock); // обработка SQL-запроса
echo '<center>';
echo 'Список журналов', '<br><br>'; // заголок WEB-страницы
echo '
<TABLE BORDER=1 CELLPADDING=2 CELLSPACING=0>
  <TR>
  <TH>код</TH>
  <TH>название</TH>
  </TR>'; // шапка таблицы
```

```
if ($R) {
$Tcol=mysql_NumRows($R);
  if ($Tcol)>0 {
  for ($i=0;$i<=$Tcol-1;$i++) {// обработка запроса по записям
        $p1=mysql_fetch_array($R);
        echo '<tr>';
        echo '<TD ALIGN=RIGHT>',$p1[0], '</TD>'; // вывод поля kod
        echo '<TD ALIGN=RIGHT>',$p1[1] '</TD>'; // вывод поля name
        echo '</tr>';
  }
  }
}
echo '</TABLE>'; // закрытие таблицы
$sock=mysql_close; // закрытие сервера
?>
</body>
</html>
```

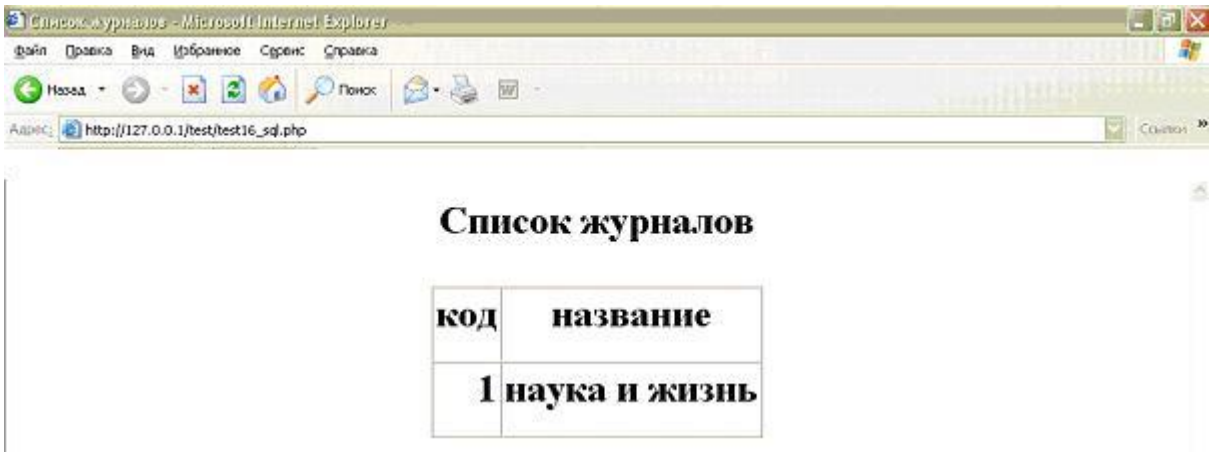

Рис. 56. Результат выполнения листинга 24

Задания к лабораторной работе № 8

Цель лабораторной работы № 8 – получить практические навыки написания *WEB*-приложений с помощью языка программирования *PHP*, используя БД *MySQL*.

1. Откройте пакет *XAMPP Control Panel*. Ярлык пакета находится на рабочем столе или в папке *C:/xampp/xampp-control.exe*. Запустите локальный *Web*-сервер *Apache* для *Windows* и *MySQL*.
2. Создайте базу данных под названием *test* (если она не создана). Создайте таблицу с заданным количеством полей. Для выполнения задания внесите 8 – 10 необходимых записей на свое усмотрение. Для создания таблицы можно использовать пакет *MySQL-Front*. *Server* – *localhost*, имя пользователя и пароль – пустые значения. Внешний вид пакета *MySQL-Front* представлен на рис. 57.

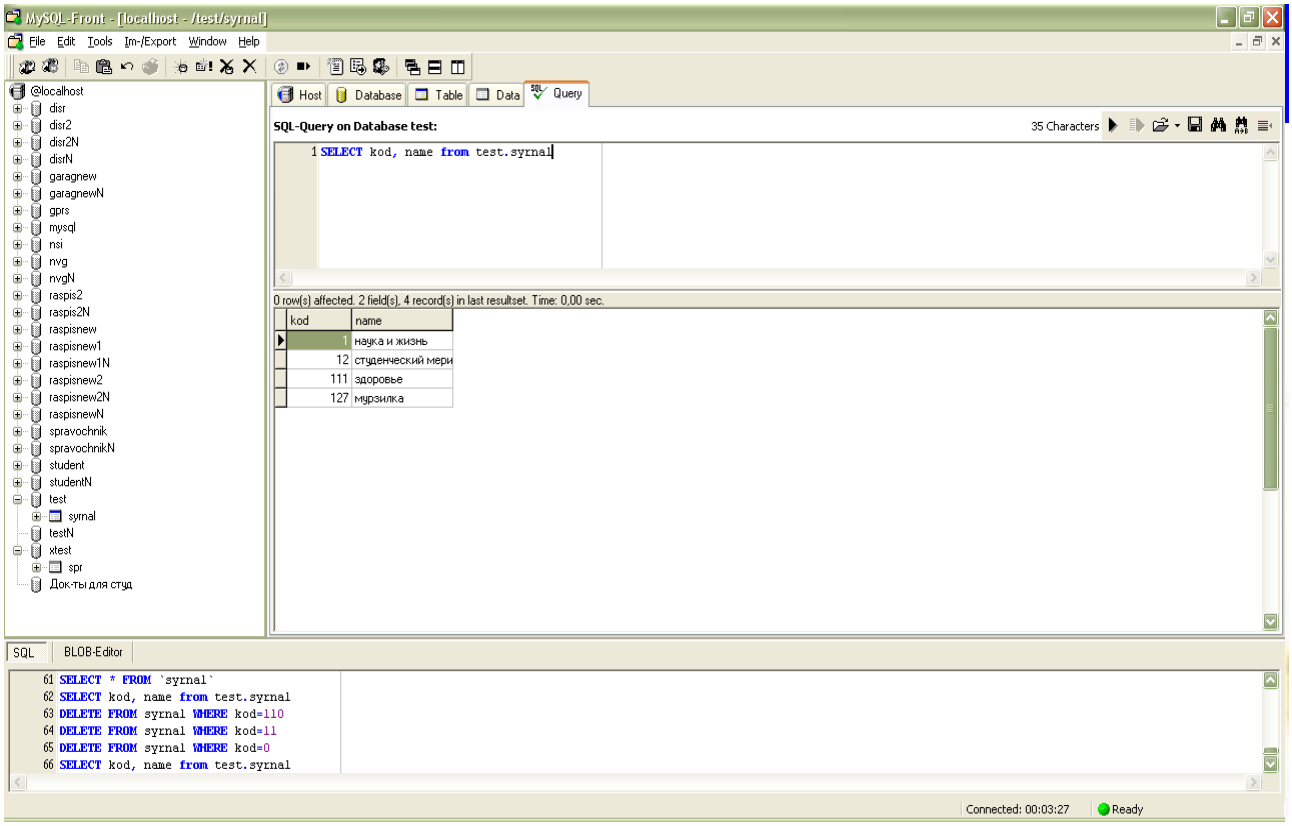

Рис. 57. Пример обработки результата запроса в приложении *MySQL-Front* в базе данных *test*

3. Используя пакет *NotePad++*, создайте первый сайт с расширением **.php*. Файл сохраните в папке, например *gruppa1* директории *C:/xampp/htdocs/*. Запустите его из адресной строки браузера, например, по доменному адресу: *http://localhost/gruppa1/primer1.php* или по *IP*-адресу: *http://127.0.0.1/gruppa1/primer1.php*. При отладке скрипта в браузере для обновления сайта используйте клавишу *F5* или кнопку *Обновить*.
4. Создайте второй сайт из предложенного задания. Все данные из полей формы *PHP* помещайте в глобальный массив *$_REQUEST*.

Отчет по лабораторной работе должен содержать:

- текст *PHP*-кода программы;
- результат выполнения программы – рисунки двух получившихся сайтов;
- выводы о проделанной работе.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Архангельский А. Я. Delphi 5 : справ. пособие / А. Я. Архангельский. М. : БИНОМ, 2001. 768 с.

Баженова И. Ю. Delphi 5: самоучитель программиста / И. Ю. Баженова. М. : КУДИЦ-ОБРАЗ, 2000. 336 с.

Бобровский С. Delphi 5 / С. Бобровский. СПб.; М. ; Харьков ; Минск : Питер, 2000. 638 с.

Гофман В .Э. Delphi 5 / В. Э. Гофман, А. Д. Хомоненко. СПб. : БХВ Санкт-Петербург, 2000. 800 с.

Грофф Дж. Энциклопедия SQL / Дж. Грофф, П. Вайнберг. 3-е изд. (+ CD). СПб. : Питер, 2003. 896 с.

Котеров Д. В. PHP 5 / Д. В. Котеров, А. Ф. Костарев. СПб. : БХВ-Петербург, 2007. 1120 с.

Тюкачев Н. Delphi 5. Создание мультимедийных приложений / Н. Тюкачев, Ю. Свиридов. СПб. ; М. ; Харьков ; Минск : Питер, 2001. 400 с.

Фаронов В. В. Delphi 5 / В. В. Фаронов. М. : Нолидж, 2001. 608 с.

DELPHI. Советы программистов / под ред. В. Озерова. 2-е изд., доп. СПб. : Символ-Плюс, 2003. 976 с.

Шкарина Л. Язык SQL / Л. Шкарина. СПб. : Питер, 2001. 592 с.

Яргер Р. MySQL и mSQL. Базы данных для небольших предприятий и Интернета / Р. Яргер, Дж. Риз, Т. Кинг. СПб. : Символ-Плюс, 2001. 560 с.

ПРИЛОЖЕНИЕ

Варианты заданий

Вариант 1

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета перемещения комплектующих деталей компьютеров (материнская плата, оперативная память, видеокарта, *CD-ROM* и т. п.) на предприятии: инвентарный номер компьютера, название и номер комплектующей детали, дата установки, рабочее место. Предприятие имеет несколько отделов, в каждом отделе несколько рабочих мест. Должности работников в разных отделах могут быть одинаковыми.
2. Создать формы для ввода справочной информации (об отделах предприятия, о рабочих местах, компьютерах, комплектующих деталях), а также текущей информации для накладной по установке деталей на ПК.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – накладная для установки деталей на ПК. Подчиненная форма – наличие комплектующих деталей на данном ПК.
4. Создать запросы, в которых необходимо вывести следующие данные:
   - инвентарный номер и наименование ПК, на которые за последний месяц установили один тип комплектующих деталей, например *CD-ROM*;
   - список работников, на ПК которых устанавливались комплектующие детали в последний месяц.
5. С помощью запроса создать таблицу «Комплектующие детали для директора». В таблицу с помощью запроса внести список комплектующих деталей, установленных на ПК директора за последний месяц.
6. Для комплектующих деталей типа *CD-ROM* обновить наименование «Устройство для чтения компакт-дисков».
7. Определить, сколько рабочих мест в каждом отделе было модернизировано (установлены комплектующие детали) за последний год.

8. Вывести информацию об установках комплектующих деталей на ПК с указанием рабочего места для любого сотрудника. Фамилия, имя, отчество работника задается в режиме диалога.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Накладная для установки КД», сгруппировав данные по инвентарному номеру ПК. Вывести комплектующие детали, дату установки КД. Отсортировать данные по номеру комплектующих деталей.

Вариант 2

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для ежедневного учета посещений врачей любой квалификации в поликлинике. В больничной карточке указывается: дата приема, фамилия, имя и отчество больного, возраст, адрес, фамилия, имя и отчество врача, поставленный диагноз, назначенная процедура, номер больничного, дата открытия и закрытия больничного.
2. Создать формы для ввода справочной информации (о враче, квалификации врача, больных, диагнозах, процедурах), а также текущей информации для больничной карточки.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – больничная карточка, подчиненная – регистрация больничного листа.
4. Создать запросы, в которых необходимо вывести данные:
   - список больных, посетивших педиатра за последний месяц с диагнозом ОРЗ;
   - список больных и процедур, назначаемых врачом за последний месяц;
   - номера больничных листов, которые были закрыты в день их открытия.
5. С помощь запроса создать таблицу «Возможная эпидемия». В таблицу с помощью запроса внести список больных, которые за последнюю неделю заболели гриппом.

6. Для больных с диагнозом ОРЗ и датой открытия больничного листа недельной давности закрыть больничный лист.
7. Определить, сколько больных в возрасте от 40 до 50 лет были на больничном в течение года с одинаковыми диагнозами.
8. Вывести информацию о всех больных, которых принимал врач. Фамилия, имя и отчество врача задается пользователем в режиме диалога.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Больничная карточка», сгруппировав данные по фамилии, имени и отчеству врача. Вывести дату приема, фамилию, имя и отчество больного, процедуры. Отсортировать данные по фамилии, имени и отчеству больного.

Вариант 3

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для диспетчера сети аптек. Должны быть указаны адрес аптеки, телефон, специализация аптеки. Информация о лекарствах: наименование лекарств, их характеристика – каких заболеваниях, доза в упаковке, доза применения, расфасовка. Информация о наличии лекарств в базе данных должна содержать: дату поступления, количество упаковок лекарства, цену, название завода-изготовителя, дату продажи последней упаковки лекарства.
2. Создать формы для ввода справочной информации (об аптеках, о лекарствах, заболеваниях), а также текущей информации о поступлении лекарств в аптеки.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – поступление лекарств в аптеки, подчиненная форма – розничная продажа лекарств.
4. Создать запросы, в которых необходимо вывести следующую информацию:
   - об аптеках, которые закупают лекарства в Венгрии;
   - о том, в каких аптеках имеется в продаже анальгин и в каких количествах;
   - лекарства, на которые не устанавливается НДС.

5. С помощью запроса создать таблицу «Наиболее дорогие лекарства». В таблицу с помощью запроса внести список лекарств от эпилепсии, которые закупали аптеки в последний год, при цене за упаковку более 1000 р.
6. Увеличить на 5 % цену продажи лекарств, поступивших в аптеку за последнюю неделю.
7. Определить сумму, которую затратила каждая аптека при покупке лекарств от гриппа за последний год.
8. Вывести информацию о лекарствах, которые покупали за последний год, в зависимости от заболевания, заданного пользователем в режиме диалога.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Поступление лекарств в аптеки», сгруппировав данные по номерам аптек. Вывести дату поступления лекарств, количество упаковок, цену упаковки, цену продажи. Отсортировать данные по наименованию лекарств.

Вариант 4

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета наличия корма в зоопарке. Информация о животных следующая: название, вид, пол, количество. Рацион питания составляется в зависимости от вида и пола животного. Пусть каждое животное кормят три раз в день, животному полагается как минимум два-три вида пищи за раз. Время кормления каждого вида животного различное.
2. Создать формы для ввода справочной информации (вид животных, рацион, продукты питания), а также текущей информации о приеме пищи животными.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – прием пищи животными, подчиненная форма – информация о животных.
4. Создать запросы, в которых необходимо вывести списки:
   - животных (вид, пол, количество), у которых в последнем месяце было усиленное питание;

- продуктов с указанием количества и даты приема пищи.

5. С помощью запроса создать таблицу «Самки-хищницы на витаминном питании». В таблицу с помощью запроса внести список животных (самок-хищников), которые последний месяц находились на витаминном питании.
6. Увеличить на 20 % количество выдаваемых продуктов, калорийность которых выше 500 ккал, для животных, находящихся на усиленном питании.
7. Определить, сколько килограммов каждого продукта за последний месяц было съедено каждый животным.
8. Вывести информацию о животных и выданных им продуктах в указанное в режиме диалога время.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Прием пищи животными», сгруппировав данные по рациону. Вывести дату, время кормления животного, название животного, название продуктов, количество продуктов. Отсортировать данные по животному.

Вариант 5

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета фондов в музее. Экспонаты музея классифицируются по видам (мебель, скульптура, книги, сувениры). Каждый зал музея имеет название, номер. Каждый экспонат имеет свой инвентарный номер, известна дата поступления его в музей, кто передал экспонат (организация или частное лицо), как (на благотворительной основе или музей приобретал за деньги), историческая ценность экспоната. Указать, выставлен ли экспонат в залах или находится в запасниках. Регистрируется дата последней реставрации, срок выставочной экспозиции до следующей реставрации.
2. Создать формы для ввода справочной информации (о залах, видах экспонатов, физических и юридических лицах, которые передавали экспонаты музею, о форме приобретения экспоната), а также текущей информации о фонде музея.

3. Создать многотабличную форму с помощью мастера форм. Главная форма – фонд музея, подчиненная форма – информация о лице, передавшем экспонат.
4. Создать запросы, в которых необходимо вывести следующую информацию:
   - список, ценность скульптур, которые находятся в запаснике и необходимо будет реставрировать в следующем году;
   - данные о физических лицах, которые продали или передали музею экспонаты;
   - список экспонатов, которые в последний год музей приобрел за деньги, причем цена приобретения соответствовала ценности экспоната.
5. С помощью запроса создать таблицу «Меценаты». В таблицу с помощью запроса внести список лиц (физических или юридических), которые безвозмездно передали музею экспонаты, ценность которых более 10 тыс. р.
6. Обновить срок следующей реставрации, заменив его на 2015 г., для мебели, которая реставрировалась в 2010 г.
7. Определить по видам, сколько экспонатов находится в запасниках и на какую сумму ценности.
8. Вывести список экспонатов ценностью выше 5 тыс. р., находящихся в зале, который задан пользователем в режиме диалога.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Фонд музея», сгруппировав экспонаты по залам. Вывести название экспоната, ценность, дату последней реставрации. Отсортировать данные по ценности экспоната. Определить ценность экспонатов в каждом зале и общую ценность всего фонда музея.

Вариант 6

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для ежедневного учета работы водителей в трамвайном депо. Учет ведется на основании документа – путевого листа. В путевом листе содержится следующая информация: дата, номер маршрута, график выхода, смена,

табельный номер водителя, фамилия, время начала работы, время окончания работы, номер вагона, простои на линии. График выхода – порядковый номер следования подвижной единицы (ПЕ) на маршруте. Простой на линии – это когда водитель и ПЕ стоят на маршрутной сети из-за сбоев в движении. Простои содержат следующую информацию: код простоя, начало простоя, окончание простоя. За водителем закреплен номер вагона и номер маршрута. Простои делятся на дорожно-транспортные происшествия, вину водителя, техническую неисправность.

2. Создать формы для ввода справочной информации (о водителе, маршруте, вагонах, простоях, графиках для каждого маршрута), а также текущей информации для путевого листа.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – путевой лист, подчиненная – информация о простоях.
4. Создать запросы, в которых необходимо вывести информацию:
   - коды простоев за последний месяц, которые произошли в первую смену;
   - список водителей, работающих на своем и на другом маршруте;
   - табельные номера водителей, работающих по две смены в день.
5. С помощью запроса создать таблицу «Опасные маршруты». В таблицу с помощью запроса внести список маршрутов, у которых за последний месяц были простои по вине дорожно-транспортных происшествий.
6. В путевом листе изменить поле «Дата», проставив текущую дату для маршрута № 2.
7. Определить, сколько простоев было по кодам простоя за последний месяц на каждом маршруте.
8. Вывести информацию о всех графиках для любого маршрута.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Путевой лист», сгруппировав данные по табельному номеру водителя. Вывести дату, номер маршрута, номер графика, номер смены, номер вагона, реальное время начала и окончания работы. Отсортировать данные по дате.

Вариант 7

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для организации компьютерного учета перемещения узлов трамвая в ходе ремонтов. В накладной необходимо учитывать следующую информацию: номер трамвая, номер узла, дату установки узла на трамвай, дату снятия узла с трамвая, причину снятия узла, фамилию слесаря, получившего задание. Для слесаря определенной квалификации оформляются отдельно две накладные (для снятия или установки узлов вагона).
2. Создать формы для ввода справочной информации (о вагоне, узле, слесаре, причинах замены узлов), а также текущей информации по накладной для установки узла на вагон и накладной для снятия узла с вагона.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – накладная для установки узла на вагон. Подчиненная – накладная для снятия узла с вагона.
4. Создать запросы, в которых необходимо вывести следующую информацию:
   - номера вагонов, в которых за последний месяц устанавливали один тип узла, например колесную пару;
   - номера узлов, которые были установлены и сняты в течение последнего квартала;
   - список слесарей, занимающихся установкой узлов в течение текущего месяца.
5. С помощью запроса создать таблицу «Часто заменяемые узлы». В таблицу с помощью запроса внести список названий узлов, снятых с вагона по причине, например, перегорания предохранителей в течение последнего квартала.
6. Обновить информацию о слесаре, который снимал детали по причине перегорания предохранителей, например: 01.01.2012.
7. Определить, сколько было замен узлов за определенный период на каждом вагоне.
8. Вывести информацию о всех вагонах, в которых произошли замены любого узла.

9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Накладная для установки узла на вагон», сгруппировав данные по номеру вагона. Вывести номер узла, дату установки узла. Отсортировать данные по номеру узла.

Вариант 8

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета расхода электроэнергии в трамвайном управлении. На предприятии имеется три подстанции. На каждой подстанции стоит по три электросчетчика. Каждый счетчик фиксирует показания соответствующего участка. Ежемесячно работник предприятия должен отчитаться в показаниях счетчика и рассчитать расход электроэнергии в денежном выражении. С 23.00 до 6.00 действует ночной тариф, также существует выходной (суббота, воскресенье) и праздничный тариф. Показания всех счетчиков фиксируются при смене тарифа.
2. Создать формы для ввода справочной информации (о подстанции, счетчиках, тарифах), а также текущей информации для учета электроэнергии.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – учетная карточка электроэнергии, подчиненная – информация о тарифах.
4. Создать запросы, в которых необходимо вывести следующие данные:
   - количество электроэнергии, которое потребила подстанция «Южная» в ночное время за последний месяц;
   - дату, когда показания счетчика не менялись, например в случае аварии на подстанции;
   - месяц максимального потребления электроэнергии подстанции «Южная» за последний год.
5. С помощью запроса создать таблицу «Наиболее потребляемые участки». В таблицу с помощью запроса внести список участков, у которых потребление

электроэнергии за последний месяц в дневное время суток составило более 1000 кВт.

6. Увеличить цену тарифа на 20 % для дневного тарифа.
7. Определить количество потраченной электроэнергии на каждом участке за последний месяц. Сгруппировать по тарифам.
8. Вывести информацию о тарифах (название, время действия, цена).
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Учетная карточка», сгруппировав данные по номеру счетчика, тарифу. Вывести дату, время, показания счетчика. Вычислить количество электроэнергии, потребляемой по каждому тарифу.

Вариант 9

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета рабочего времени сотрудников, т. е. организовать табель рабочего времени. Необходимо организовать учет следующих данных: табельный номер сотрудника, фамилия, имя, отчество, дата и время работы, причина невыхода на работу, адресные данные сотрудников, телефон, возраст, стаж работы. Также необходимо проверять соответствие режима работы каждого сотрудника его индивидуальному графику работы.
2. Создать формы для ввода справочной информации (о сотрудниках, режимах работы, причинах невыхода на работу), а также текущей информации для заполнения табельного листа.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – табельный лист, подчиненная – сведения о сотруднике.
4. Создать запросы, в которых необходимо вывести список сотрудников:
   - которые не вышли на работу по причине отгулов за последний месяц;
   - указав, в каком режиме они работали 10-го числа последнего месяца;
   - у которых поменялся режим работы.

5. С помощью запроса создать таблицу «Болеющие сотрудники». В таблицу с помощью запроса внести список сотрудников, которые в последний месяц не выходили на работу по причине болезни.
6. Для ночных смен обновить время режима работы с 8 на 6 часов.
7. Определить, сколько времени работал каждый сотрудник в последнем месяце.
8. Вывести информацию о сотрудниках (табельный номер, фамилия, имя, отчество, адрес, режим работы), которые за последний год, например, не выходили на работу.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Табельный лист», сгруппировав данные по табельному номеру водителя. Вывести дату, режим работы, время режима работы, время работы (реальное), причину невыхода на работу (если таковая есть). Отсортировать данные по дате.

Вариант 10

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для ежедневного учета движения библиотечного фонда: автор и название книги, год издания, фамилия, имя, отчество читателя, причина отсутствия книги в библиотеке. Книги необходимо учитывать по категориям. Читательская карточка должна содержать: фамилию, имя, отчество читателя, адрес, место работы, телефон рабочий, телефон домашний. В случае задержки книги читателем принимаются меры, которые фиксируются в читательской карточке.
2. Создать формы для ввода справочной информации (о категориях книг, книгах, читателе, мерах, принимаемых по отношению к должникам), а также текущей информации «Заказ читателя», «Возврат книги».
3. Создать многотабличную форму с помощью мастера форм. Главная форма – заказ читателя, подчиненная – возврат книги.

4. Создать запросы, в которых необходимо вывести следующие списки:
   - книг (авторы, название, год издания) категории «детективы», которые заказывали читатели за последний месяц;
   - читателей, заказавших книги в прошлом месяце. Указать, возвратили или нет книгу;
   - книг, которые запрашивал читальный зал (дата возврата соответствует дате заказа) в последний месяц.
5. С помощью запроса создать таблицу «Злостные должники». В таблицу с помощью запроса внести список читателей (фамилия, имя, отчество, адрес, телефон), которые не сдали книги после второго телефонного звонка.
6. Присвоить читателю имя «должник», если книга не возвращена на следующий день после установленной даты возврата.
7. Определить, сколько должников не вернули книги каждой категории.
8. Вывести список возвращенных книг (авторы, название книги, год издания).
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Заказ читателя», сгруппировав категории книг. Вывести дату заказа, фамилия, имя, отчество читателя, автора, название книги, год издания. Отсортировать данные по фамилии, имени, отчеству читателя.

Вариант 11

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для учета дорожно-транспортных происшествий (ДТП) в городе для ГИБДД. Учитывать только ДТП, когда автомобиль сбивает пешехода. При организации учета ДТП необходимо заносить данные: дату, время, фамилию, имя и отчество шофера, фамилию, имя и отчество пешехода, фамилию, имя и отчество свидетелей, тяжесть последствий (без травм, легкая, тяжелая травма, смертельный случай), виновность участников ДТП, адреса участников ДТП.

2. Создать формы для ввода справочной информации с помощью мастера форм по справочникам: о машине, пострадавших, виновности, тяжести последствия, свидетелях. Создать форму «Учет ДТП».
3. Создать многотабличную форму с помощью мастера форм. Главная форма – учетная карточка ДТП, подчиненная форма – информация о свидетелях.
4. Создать запросы, в которых необходимо вывести следующие данные:
   - информацию о машинах, которые за последний месяц сбили пешехода на перекрестке Ленина - Луначарского;
   - список свидетелей аварий, которые происходили в течение последних суток и закончились тяжелой травмой для пешехода;
   - вывести номера аварий, у которых не менее двух свидетелей.
5. С помощью запроса создать таблицу «Опасные перекрестки». В таблицу с помощью запроса внести список перекрестков, на которых за последний месяц произошли аварии одного типа, например сбит пешеход по вине неработающего светофора.
6. Изменить адрес водителей: Свердловск на Екатеринбург.
7. Определить, сколько аварий было по видам виновности участников ДТП за последний год.
8. Вывести информацию о пострадавших в зависимости от тяжести последствия.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Учет ДТП», сгруппировав данные по виновности участников ДТП. Вывести дату аварии, номер машины, фамилию, имя и отчество водителя, фамилию, имя и отчество пострадавшего, место аварии, тяжести последствия ДТП. Отсортировать данные по степени последствия ДТП.

Вариант 12

1. Спроектировать базу данных в первой, второй и третьей нормальных формах для заказа билета на самолет, в которой указать дату и время заказа билета,

рейс, место, дату и время отправления, дату и время прилета, фамилию, имя и отчество пассажира, номер паспорта, цену билета. Рейс может быть международным, местным, чартерным, регулярным. Число мест в самолете зависит от модели самолета.

2. Создать формы для ввода справочной информации (о номерах самолетов, моделях самолетов, рейсах, категории рейсов, пассажирах), а также текущей информации о заказах билетов.
3. Создать многотабличную форму с помощью мастера форм. Главная форма – заказ билетов, подчиненная форма – информация о рейсе.
4. Создать запросы, в которых необходимо вывести списки:
   - пассажиров, заказавших билеты вчера на завтрашние рейсы в Москву;
   - пассажиров, заказавших билеты в день отлета.
5. С помощью запроса создать таблицу «Чартерные рейсы», в которой указать список пассажиров, отлетающих сегодня на чартерных рейсах на ТУ-154.
6. Обновить цену билетов, увеличив ее на 20 % с 1 числа следующего месяца.
7. Определить, сколько было заказано билетов по категориям рейсов за последний месяц.
8. Вывести информацию о рейсах, отправляющихся в ближайший месяц в город, который задан пользователем в режиме диалога.
9. С помощью компонентов библиотеки *HTTPApp* (страница *Internet* среды *Delphi*) создать *HTML*-страницу в браузере на основе таблицы «Заказ билетов», сгруппировав данные по категории рейсов. Вывести дату и время заказа, дату и время отлета, дату и время прилета, фамилию, имя и отчество пассажира. Отсортировать данные по категориям рейсам.

ОГЛАВЛЕНИЕ

Учебное издание

Дружинина Надежда Геннадьевна

Трофимова Ольга Геннадиевна

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Редактор *И. В. Меркурьева*

Компьютерный набор *О. Г. Трофимовой, Н. Г. Дружининой*

Подписано в печать 20.11.2012. Формат 60x84 1/16.

Бумага писчая. Плоская печать. Усл. печ. л. 8,37.

Уч.-изд. л. 6,3. Тираж 50 экз. Заказ №2379.

Редакционно-издательский отдел УрФУ

620002, Екатеринбург, ул. Мира, 19

E-mail: rio@ustu.ru

Отпечатано в типографии Издательско-полиграфического центра УрФУ

620000, Екатеринбург, ул. Тургенева, 4

Тел. +7(343) 350-56-64, 350-90-13

Факс: +7(343) 358-93-06

E-mail: press.info@ustu.ru